A New Lower Bound for the List Update Problem in the Partial Cost Model

June 21, 1999

Christoph Ambühl[1], Bernd Gärtner[1], Bernhard von Stengel[2]

Abstract. The optimal competitive ratio for a randomized online list update algorithm is known to be at least 1.5 and at most 1.6, but the remaining gap is not yet closed. We present a new lower bound of 1.50084 for the partial cost model. The construction is based on game trees with incomplete information, which seem to be generally useful for the competitive analysis of online algorithms.

Keywords. On-line algorithms, analysis of algorithms, competitive analysis, list-update.

1. Introduction

The *list update problem* is a classical online problem in the area of self-organizing data structures [4]. Requests to items in an unsorted linear list must be served while maintaining the list so that access costs remain small. We assume the *partial cost model* where accessing the ith item in the list incurs a cost of $i-1$ units. This is simpler to analyze than the original *full cost model* [14] where that cost is i. After an item has been requested, it may be moved free of charge closer to the front of the list. This is called a *free exchange*. Any other exchange of two consecutive items in the list incurs cost one and is called a *paid exchange*.

An *online* algorithm must serve the sequence σ of requests one item at a time, without knowledge of future requests. An optimum *offline* algorithm knows the entire sequence σ in advance and can serve it with minimum cost $OFF(\sigma)$. If the online algorithm serves σ with cost $ON(\sigma)$, then it is called *c-competitive* if for a suitable constant b,

$$ON(\sigma) \leq c \cdot OFF(\sigma) + b \tag{1}$$

for all request sequences σ. The *competitive ratio* c in this inequality is the standard yardstick for measuring the performance of the online algorithm. The well-known *move-to-front* rule MTF, for example, which moves each item to the front of the list after it has been requested, is 2-competitive [14, 15]. This is also the best possible competitiveness for any deterministic online algorithm for the list update problem [10].

[1]Institute for Theoretical Computer Science, ETH Zürich, 8092 Zürich, Switzerland. Email: ambuehl@inf.ethz.ch, gaertner@inf.ethz.ch

[2]Mathematics Department, London School of Economics, London WC2A 2AE, Great Britain. Email: stengel@maths.lse.ac.uk. Support by a Heisenberg grant from the Deutsche Forschungsgemeinschaft and the hospitality of the ETH Zürich for this research are gratefully acknowledged.

Randomized algorithms can perform better on average [9]. Such an algorithm is called c-competitive if

$$E\big[ON(\sigma)\big] \leq c \cdot OFF(\sigma) + b \tag{2}$$

where the expectation is taken over the randomized choices of the online algorithm. Furthermore, we call the algorithm *strictly c-competitive* if (2) holds with $b = 0$.

The best randomized list update algorithm known to date is 1.6-competitive. This algorithm *COMB* [2] serves the request sequence with probability 4/5 using the algorithm *BIT* [14], which alternately moves a requested item to the front or leaves it in place. With probability 1/5, *COMB* treats the request sequence using a deterministic *TIMESTAMP* algorithm [1], where a requested item x is moved in front of the first item in the list that has been requested at most once since the last request to x.

Randomization is useful only against the *oblivious adversary* [5] that generates request sequences without observing the randomized choices of the online algorithm. If the adversary can observe those choices, it can generate requests as if the algorithm was deterministic, which is then at best 2-competitive. We therefore consider only the interesting situation of the oblivious adversary. Lower bounds for the competitive ratio can be proved using Yao's theorem [18]: If there is a probability distribution on request sequences so that the resulting *expected competitive ratio* for any deterministic online algorithm is d or higher, then every deterministic or randomized online algorithm has competitive ratio d or higher [8]. Teia [16] described a simple distribution on request sequences that, adapted to the partial cost model, shows a lower bound of 1.5. The optimal competitive ratio for the list update problem is therefore between 1.5 and 1.6, but the true value is as yet unknown.

For lists with up to four items, it is possible to construct an online list update algorithm that is 1.5-competitive [3] and therefore optimal. In this paper, we show a lower bound that is greater than 1.5 when the list has at least five items. We will prove this bound for the standard assumption that algorithms may use paid exchanges. One can also prove a lower bound above 1.5 for the variant of the list update problem where only free exchanges are allowed. For that purpose, we have to modify and extend our method in certain ways, as mentioned at the end of this paper.

Our construction uses a *game tree* where alternately the adversary generates a request and the online algorithm serves it. The adversary is not informed about the action of the online algorithm, so the game tree has *imperfect information* [12]. We consider a finite tree where after some requests, the ratio of online versus optimal offline cost is the payoff to the adversary. This defines a zero-sum game, which we solve by linear programming. For a game tree that is sufficiently deep, and restricted to a suitable subset of requests so that the tree is not too large in order to stay solvable, this game has a value of more than 1.50084. This shows that any strictly c-competitive online algorithm fulfills $c \geq 1.50084$. In order to derive from this a new lower bound for the competitive ratio c according to (1) with a nonzero constant b, one has to generate arbitrarily long request sequences. This can be achieved by composing the game trees repetitively, as we will show.

A drawback is our assumption of the partial instead of the full cost model. In the latter model, where a request to the ith item in the list incurs cost i, the known lower bound is $1.5-5/(n+5)$ for a list with n items. This result by Teia [16] yields a lower bound for the competitive ratio much below 1.5 when the list is short. In fact, the algorithm *COMB* [2] is 1.5-competitive when $n < 9$. To prove a lower bound above 1.5 for the full cost model we would have to extend our construction to longer lists. Unfortunately, a straightforward extension cannot compensate for the reduction of the competitive ratio by $5/(n+5)$ (or any term proportional to $1/n$) when considering the full instead of the partial cost model, so this case remains open. Nevertheless, we think a result for the partial cost model is still interesting since that model is more canonical when one looks at the analysis, and it is still close to the original problem formulation.

2. Pairwise analysis and partial orders

The analysis of a list update algorithm is greatly simplified by observing separately the relative movement of any pair of items in the list. Let σ be a sequence of requests. Consider any deterministic algorithm A that processes σ. For any two items x and y in the list, let $A_{xy}(\sigma)$ be the number of times where y is requested and is behind x in the list, or vice versa. Then it is easy to show [6, 9, 2] that

$$A(\sigma) = \sum_{\{x,y\} \subseteq L \,:\, x \neq y} A_{xy}(\sigma)\,,$$

where L is the set of items in the list. In that way, $A_{xy}(\sigma)$ represents the cost of the online algorithm *projected* to the unordered pair $\{x, y\}$ of items.

Let σ_{xy} be the request sequence σ with all items other than x or y deleted. Many list update algorithms, like *MTF*, *BIT*, and *TIMESTAMP*, are *projective* in the sense that at any time the relative order of two items x and y in the list depends only on the projected request sequence σ_{xy} and the initial order of x and y, which we denote by $[xy]$ if x precedes y. (In general, we list items between square brackets to denote their current order in the list maintained by the algorithm.)

For the optimal offline algorithm *OFF*, the projected cost $OFF_{xy}(\sigma)$ is clearly at least as high as the cost of serving σ_{xy} optimally on the two-element list consisting of x and y. The latter cost is easy to determine since, for example, it is always optimal to move an item to the front at the first of two or more successive requests. In fact, the item *must* be moved to the front at the first of three or more successive requests. On the other hand, it is usually not optimal to move an item that is requested only once. Hence, for any two items x and y where x precedes y, an online algorithm serving a request to y can either leave y behind x or move y in front of x, which, either way, is a "mistake" depending on whether y will be requested again before x or not.

Based on this observation, Teia [16] has constructed a lower bound of 1.5 for the competitive ratio. The requests are generated in *runs* which are repeated indefinitely. At the start of each run, the list currently maintained by the offline algorithm has a particular order $[x_1 x_2 \ldots x_n]$. Then this list is traversed from front to back, requesting each item with equal probability either once or three times. If an item is requested three times, then it is moved to the front at the first request, otherwise left in place, which is an optimal offline treatment. This results in a new offline list, which determines the next run.

The following table (3) lists the resulting costs for the possible actions of the online algorithm, projected on items x and y. In that table, *WAIT* refers to an online algorithm that moves the requested item only at the second request in succession (if it is not moved then, the online costs are even higher), and *MTF* moves the item to the front at the first request. In (3), item x is assumed to precede y in the lists maintained by both offline and online algorithm. The four request sequences each have probability 1/4. For each of the four possible combinations of *WAIT* and *MTF*, the column *ON* denotes the online cost and I_{after} denotes the number of *inversions* in the online list after the requests have been served. An inversion is a transposition of two items relative to their position in the offline list.

σ_{xy}	*OFF* with $[xy]$	*ON* with $[xy]$, $I_{\text{before}} = 0$							
		x *WAIT* y *WAIT*		x *MTF* y *WAIT*		x *WAIT* y *MTF*		x *MTF* y *MTF*	
		ON	I_{after}	*ON*	I_{after}	*ON*	I_{after}	*ON*	I_{after}
xy	1	1	0	1	0	1	1	1	1
$xxxy$	1	1	0	1	0	1	1	1	1
$xyyy$	1	2	0	2	0	1	0	1	0
$xxxyyy$	1	2	0	2	0	1	0	1	0
$4 \cdot E[ON + \Delta I]$		$6+0$		$6+0$		$4+2$		$4+2$	

(3)

Without inversions, the *MTF* algorithm, for example, would incur the same cost as the optimal offline cost. However, the inversion increases the online cost by a full unit in the next run, where $[xy]$ is the order for the offline algorithm but $[yx]$ is the order of x and y in the list used by the online algorithm. The following table shows these online costs when the algorithm starts with such an inversion, denoted by $I_{\text{before}} = 1$.

σ_{xy}	*OFF* with $[xy]$	*ON* with inversion $[yx]$, $I_{\text{before}} = 1$							
		x *WAIT* y *WAIT*		x *MTF* y *WAIT*		x *WAIT* y *MTF*		x *MTF* y *MTF*	
		ON	I_{after}	*ON*	I_{after}	*ON*	I_{after}	*ON*	I_{after}
xy	1	1	1	2	0	1	1	2	1
$xxxy$	1	3	0	2	0	3	1	2	1
$xyyy$	1	1	0	3	0	1	0	2	0
$xxxyyy$	1	4	0	3	0	3	0	2	0
$4 \cdot E[ON + \Delta I\,]$		$9-3$		$10-4$		$8-2$		$8-2$	

(4)

Tables (3) and (4) list all possible online actions, except for those that perform even worse (leaving a triply requested item in place, for example). Note that it does not matter if the online algorithm conditions its action on the presence of inversions or not.

Let T be the distribution on request sequences generated by the described method of Teia. Then the expected online costs together with the change in the number of inversions fulfill the inequality

$$E[ON(T) + \Delta I\,] = E[ON(T) - I_{\text{before}} + I_{\text{after}}] \geq 1.5\, OFF(T)\,. \tag{5}$$

This follows from (3) and (4) by considering the projected sequences and telescoping the sum for the inversion counts from one run to the next (where I_{before} for that run is equal to I_{after} for the previous run and cancels). Inequality (5) shows that the number of inversions can serve as a *potential function* [7]. The variation $\Delta I = I_{\text{after}} - I_{\text{before}}$ of this potential function is bounded, so that (5) implies that any online algorithm is at most 1.5-competitive for the distribution T on request sequences. We will extend Teia's method in our lower bound construction.

Using *partial orders*, one can construct a 1.5-competitive list update algorithm for lists with up to four items [3]. The partial order is initially equal to the linear order of the items in the list. After each request, the partial order is modified as follows, where $x\|y$ means that x and y are incomparable:

partial order before	after request to		
	$z \notin \{x, y\}$	x	y
$x\|y$	$x\|y$	$x < y$	$y < x$
$x < y$	$x < y$	$x < y$	$x\|y$

That is, a request only affects the requested item y in relation to the remaining items. Then y is in front of all items x except if $x < y$ held before, which is changed to $x\|y$. The initial order in the list and the request sequence determine the resulting partial order. One can generate an arbitrary partial order in this way [3].

The partial order defines a *position* $p(x) = |\{y \mid y < x\}| + |\{y \mid y \| x\}|/2$ for each item x. If the online algorithm can maintain a distribution on lists so that the expected cost of accessing an item x is equal to $p(x)$, then this algorithm is 1.5-competitive [3]. One can show that then x is with probability one behind all items y so that $y < x$, and precedes with probability 1/2 those items y where $x \| y$. Incomparable elements reflect the possibility of a "mistake" of not transposing these items, which should have probability 1/2. For lists with up to four items, one can maintain such a distribution using two lists only. That is, the partial order is represented as the intersection of two lists, where each list is updated by moving the requested item suitably to the front, using only free exchanges. The algorithm works by choosing one of these lists at the beginning with probability 1/2 as the actual list and serving it so as to maintain the partial order (with the aid of the separately stored second list).

The partial order approach is very natural for the projection on pairs and when the online algorithm can only use free exchanges. A lower bound above 1.5 must exploit a failure of this algorithm. This is already possible for lists with five items, despite the fact that all five-element partial orders are two-dimensional (representable as the intersection of two linear orders). Namely, let the items be integers and let the initial list be [12345], and consider the request sequences

$$\sigma_1 = 4254 \qquad \text{and} \qquad \sigma_2 = 4253\,. \tag{6}$$

After the first request to 4, the partial order states $4\|1$, $4\|2$, $4\|3$, and $4 < 5$, and otherwise $1 < 2 < 3 < 5$. Using a free exchange, 4 can only be moved forward and has to precede 3, 2, 1 each with probability 1/2. This is achieved uniquely with the uniform distribution on the two lists [12345] and [41235] (this, as well as the following, holds even though distributions on more than two lists are allowed). The next request to 2 induces $2 < 4$, so 2 must be moved in front of 4 in the list [41235], where 2 already passes 1, which yields the unique uniform distribution on [12345] and [24135]. The next request to 5 entails that 5 is incomparable with all other items. It can be handled deterministically in exactly two ways (or by a random choice between these two ways): Either 5 is moved to the front in [24135], yielding the two lists [12345] and [52413] with equal probability, or 5 is moved to the front in [12345], yielding the two lists [51234] and [24135] with equal probability. If the two lists are [12345] and [52413], the algorithm must disagree with the partial order after the request to 4 as in σ_1, since then 4 must precede both 1 and 5 in both lists (so 4 is moved to the front in both lists) but then incorrectly passes 2 where only $2\|4$ should hold. Similarly, for the two lists [51234] and [24135] the request to 3 as in σ_2 moves 3 in front of 5 and 4 in both lists, so that it passes 1, violating $1\|3$. Thus, either σ_1 or σ_2 in (6) causes the poset-based algorithm to fail, which otherwise achieves a competitive ratio of 1.5. These sequences will be used with certain probabilities in our lower bound construction.

3. Game trees with imperfect information

Competitive analysis can be phrased as a zero-sum game between two players, the adversary and the online algorithm (or *online player*). In order to deal with finite games, we assume a finite set S of request sequences σ (of a given bounded length, for example), which represent the pure strategies of the adversary. These can be *mixed* by randomization. The online player has a finite number N of possible ways of deterministically serving these request sequences. These deterministic online algorithms can also be chosen randomly by suitable probabilities p_j for $1 \leq j \leq N$. In this context of finitely many request sequences, an arbitrary constant b in (2) is not reasonable, so we look at strict competitiveness. The randomized online algorithm is strictly c-competitive if for all σ in S,

$$\sum_{j=1}^{N} p_j \, ON_j(\sigma) \leq c \cdot OFF(\sigma)\,, \tag{7}$$

where $ON_j(\sigma)$ is the cost incurred by the jth online algorithm and $OFF(\sigma)$ is the optimal offline cost for serving σ. We can disregard the trivial sequences σ with $OFF(\sigma) = 0$ that consist only of requests to the first item in the list. In this case (7) is equivalent to

$$\sum_{j=1}^{N} p_j \, \frac{ON_j(\sigma)}{OFF(\sigma)} \leq c\,. \tag{8}$$

The terms $ON_j(\sigma)/OFF(\sigma)$ in (8), for $1 \leq j \leq N$ and $\sigma \in S$, can be treated as a payoff to the adversary in a zero-sum game matrix with rows σ and columns j. Correspondingly, a lower bound d for the strict competitive ratio is an expected competitive ratio [8] resulting from a distribution on request sequences. This distribution is a mixed strategy of the adversary with probabilities q_σ for σ in S so that for all online strategies $j = 1, \ldots, N$

$$\sum_{\sigma \in S} q_\sigma \, \frac{ON_j(\sigma)}{OFF(\sigma)} \geq d\,. \tag{9}$$

The minimax theorem for zero-sum games [18] asserts that there are mixed strategies for both players and reals c and d so that (8) and (9) hold with $d = c$. Then c is the "value" of the game and the optimal strict competitive ratio for the chosen finite approximation of the list update problem. Note that it depends on the admitted length of request sequences. Due to the complicated implicit definition and large size of the game matrix, we only know bounds c and d in (8) and (9) that hold irrespective of the length of the request sequences where $d < c$.

The number of request sequences is exponential in the length of the sequences. The online player has an even larger number of strategies since that player's actions are conditional on the observed requests. This is best described by a *game tree*. At each nonterminal node of the tree, a player makes a move corresponding to an outgoing edge. The game starts at the root of the tree where the adversary chooses the first request. Then, the online player moves with actions corresponding to the possible reorderings of the list after the request. There are $n!$ actions corresponding to all possible reorderings. (Later, we will see that most of them need not be considered.) The players continue to move alternatingly until the last request and the reaction by the online player. Each leaf of the tree defines a sequence σ and an online cost $ON(\sigma)$ (depending on the online actions leading to that leaf), with payoff $ON(\sigma)/OFF(\sigma)$ to the adversary.

The restricted information of the adversary in this game tree is modeled by *information sets* [12]. Here, an information set is a set of nodes where the adversary is to move and which are preceded by the same previous moves of the adversary himself. Hence, the nodes in the set differ only by the preceding moves of the online player, which the adversary cannot observe. An action of the adversary is assigned to each information set (rather than an individual node) and is by the definition the same action for every node in that set. On the other hand, the online player is fully informed about past requests, so his information sets are singletons. Figure 1 shows the initial part of the game tree for a list with three items for the first and second request by the adversary, and the first online response, here restricted to free exchanges only.

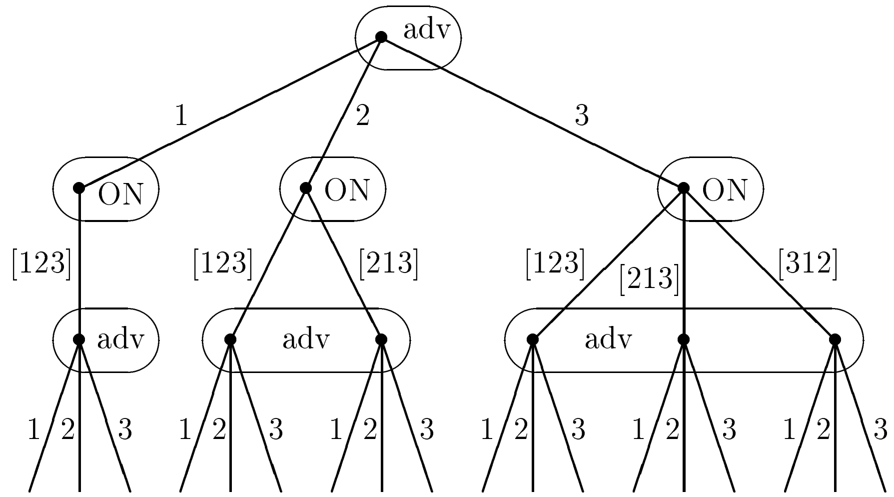

Figure 1. Game tree with information sets.

A pure *strategy* in a game tree assigns a move to every information set of a player, except for those that are unreachable due to an earlier choice of that player. Here, the online player has information sets (like in Figure 1) where each combination of moves defines a different strategy. This induces an *exponential* growth of the number of strategies in the *size of the tree*. The strategic approach using a game matrix as in (8) and (9) becomes therefore computationally intractable even if the game tree is still of moderate size. Instead, we have used a recent method [17, 11] which allows to solve a game tree with a "sequence form" game matrix and corresponding linear program that has the *same* size as the game tree.

Using game trees, a first approach to finding a randomized strategy for the adversary is the following. Consider a list with five items, the minimum number where a competitive ratio above 1.5 is possible. Fix a maximum length m of request sequences, and generate the game tree for requests up to that length. At each leaf, the payoff to the adversary is the quotient of online and offline cost for serving that sequence. Then convert the game tree to a linear program, and compute optimal strategies with an LP solver (we used CPLEX).

However, this straightforward method does not lead to a strict competitiveness above 1.5, for two reasons. First, "mistakes" of an algorithm, like in the last column in (3), manifest themselves only later as actual costs, so there is little hope for an improved lower bound using short request sequences. Secondly, even if only short sequences are considered, the online player has $n!$ responses to every move of the adversary, so that the game tree grows so fast that the LP becomes computationally infeasible already for very small values of m.

The first problem is overcome by adding the number of *inversions* of the online list, denoted by I_{after} in (3) and (4) above, to the payoff at each leaf. This yields a strict competitive ratio greater than 1.5 for rather short sequences. The inversions are converted into actual costs by attaching a "gadget" to each leaf of the game tree that generates requests similar to Teia's lower bound construction. The next section describes the details.

The second problem, the extremely rapid growth of the game tree, is avoided as follows. First, we limit the possible moves of the online player by allowing only paid exchanges of a special form, so-called *subset transfers* [13]. A subset transfer chooses some items in front of the requested item x and puts them in the same order directly behind x (e.g. $[1\underline{2}3\underline{45}x67] \to [13x\underline{245}67]$). Afterwards, the adversary's strategy computed against this "weak" online player is tested against *all* deterministic strategies of the online player, which can be done quickly by dynamic programming. Then the lower bound still holds, that is, the "strong" online player who may use arbitrary paid exchanges cannot profit from its additional power.

4. The game tree gadgets

We compose a game tree from two types of trees or "gadgets". The first gadget called FLUP (for "finite list update problem") has a small, irregular structure. The second gadget called IC (for "inversion converter") is regularly structured. Both gadgets come with a randomized strategy for the adversary, which has been computed by linear programming for FLUP. An instance of IC is appended to each leaf of FLUP. The resulting tree with the specified strategy of the adversary defines a one-player decision problem for the online player that has an expected strict competitive ratio of at least $1.5 + 1/1184$, about 1.50084, for the simplest version of FLUP that we found; larger versions of FLUP give higher lower bounds.

Both gadgets assume a particular state of the offline list, which is a parameter that determines the adversary strategy. Furthermore, at the root of FLUP (which is the beginning of the entire game), it is assumed that both online and offline list are in the same state, say [12345]. Then the adversary strategy for FLUP generates only the request sequences 4, 425, 4253, and 4254 with positive probability, which are the sequences in (6) or a prefix thereof. After the responses of the online player to one of these request sequences, the FLUP tree terminates in a leaf with a particular status of the online list and of the offline list, where the latter is *also chosen by the adversary*, independently of the online list. For the request sequence 4, that offline list is [41235], that is, the offline algorithm has moved 4 to the front. If the FLUP game terminates after the request sequence 425, the adversary makes an additional *internal* choice, unobserved by the online player, between the offline lists [51234] and [52134]. In the first case, the offline player brought 5 to the front but left 4 and 2 in their place, in the second, 2 was also moved to the front. Similar choices are performed between the offline lists for the request sequences 4253 and 4254.

requests	offline list	probability	OFF	$MTF+I$	$WAIT+I$
4	[41235]	396/1184	3	$3+0$	$3+3$
425	[51234]	114/1184	8	$9+3$	$8+4$
425	[52134]	38/1184	8	$9+2$	$8+5$
4253	[12345]	40/1184	10	$13+7$	$10+0$
4253	[13524]	240/1184	10	$13+4$	$10+3$
4253	[23451]	200/1184	10	$13+3$	$10+4$
4254	[24513]	117/1184	9	$11+2$	$11+4$
4254	[41523]	39/1184	9	$11+2$	$11+2$

(10)

The specific probabilities for these choices of the adversary in FLUP are shown in (10). The last three columns denote the cost for the offline algorithm and, as an example, the two online algorithms MTF and $WAIT$, where $WAIT$ moves an item to the front at the second request. The FLUP tree starts with 4 as the first request, followed by the possible responses of the online player. Next, the adversary exits with probability 396/1184, without a request, to the leaf with offline list [41235], and with complementary probability requests item 2, which is followed by the online move, and so on.

Each leaf of the FLUP tree is the root of an IC gadget which generates requests (similar to the runs in Teia's construction, see below), depending on the offline list. The number of inversions of the online list relative to this offline list is denoted by I in (10). The purpose of the IC gadget is to convert these inversions into actual costs. Any request sequence generated by the IC gadget can be treated with the same offline cost v, here $v = 30$. Thereby, the online algorithm makes mistakes relative to the offline algorithm, so that the additional online cost in IC is $1.5v$.

Let ON be the online cost incurred inside FLUP. Then FLUP can be represented as a game tree where the IC gadget at each leaf is replaced by the payoff D to the adversary,

$$D = \frac{ON + I + 1.5v}{OFF + v}\,. \tag{11}$$

Using these payoffs, the probabilities in (10) have been computed by linear programming. One can show that any online strategy, as represented in the FLUP tree, has an expected strict competitive ratio of at least $1.5 + \frac{1}{1184}$, or about 1.50084. Two optimal online strategies are *MTF* and *WAIT*, where the values of $ON + I$ as used in (11) are also shown in (10). Here, *WAIT* moves only item 4 to the front at the end of the request sequence 4254.

The well-defined behavior of the IC gadget allows to replace it by a single payoff D as in (11). Furthermore, the online player knows that the FLUP gadget has been left and the IC gadget has been entered, since IC starts with a request to the first item in the offline list, like 4 when that list is [41235] as in the first row of (10). In this context, we make a certain assumption about the internal choice of the adversary between different offline lists. Namely, at the start of the IC gadgets for the offline lists [12345] and [13524] which follow the request sequence 4253, the first request is 1 and then the online player cannot yet tell which IC gadget has been entered. Strictly speaking, the two gadgets have to be bridged by appropriate information sets for the online player. However, we assume instead that the internal choice of the adversary between the two lists is *revealed* to the online player at the beginning of IC, which is implicit in replacing IC by a single payoff. This is allowed since it merely weakens the position of the adversary: Any online strategy without this extra information can also be used when the online player is informed about the adversary's internal choice, so then the online payoff cannot be worse.

The offline list assigned to a leaf of the FLUP gadget is part of an optimal offline treatment (computed similar to [13]) for the entire request sequence. However, that list may even be part of a suboptimal offline treatment, which suffices for showing a lower bound since it merely increases the denominator in (9). Some of the offline costs in (10) can only be realized with *paid exchanges* by the offline algorithm. For example, the requests 4253 are served with cost 10 yielding the offline list [23451] by initial paid exchanges that move 1 to the end of the list. With free exchanges, this can only be achieved by moving every requested item in front of 1, which would result in higher costs.

In the remainder of this section, we describe the IC gadget. Its purpose is to convert the inversions at the end of the FLUP game to real costs while maintaining the lower bound of at least 1.5. At the same time, these inversions are destroyed so that both the online list and the offline list are in the same order after serving the IC.

The IC extends the construction by Teia [16] described in Section 2 above. Let T_k be the sequence that requests the first k items of the current offline list in ascending order, requesting each item with probability 1/2 either once or three times. Assume that the offline algorithm treats T_k by moving an item that is requested three times to the front at the first request, leaving any other item in place, which is optimal. The triply requested items, in reverse order, are then the first items of the new offline list, followed by the remaining items in the order they had before. Then T_n is a run as used in Teia's construction for a list with n items. The random request sequence generated there can be written as T_n^w, that is, a w-fold repetition of T_n, where w goes to infinity. Note that the offline list and hence the order of the requests *changes* from one run T_n to the next, so T_n^2, for example, is *not* a repetition of two identical sequences. The optimal offline treatment of T_k costs $\binom{k}{2}$ units.

The difference between our construction and Teia's is the use of only a prefix of the elements in the offline list. We show

$$E[\,ON(T_k) - I_{\text{before}} + I_{\text{after}}] \geq 1.5\; OFF(T_k)\,, \tag{12}$$

which for $k = n$ has already been proved above, see (5). To see (12) also for $k < n$, we use projection on pairs and consider the case of only two items. If none of the two items occur in T_k, both sides of (12) are zero, because, by definition, projection on pairs ignores items that are not projected. If only one item occurs in T_k, only the first one in the offline list was requested, so $OFF(T_k) = 0$. Furthermore, $ON(T_k) - I_{\text{before}} \geq 0$, because the online algorithm incurs cost at least one if there is an inversion. This shows (12).

As in (5) above, (12) can be extended to concatenations T of sequences T_k. We let IC be the randomly generated sequence defined by

$$IC := T_4^3 \quad T_3^3 \quad T_2^3 \quad T_1^3,$$

which by the preceding considerations fulfills

$$E[\,ON(IC)] \geq 1.5\; OFF(IC) + I_{\text{before}} - E[I_{\text{after}}]\,. \tag{13}$$

If $E[I_{\text{after}}] = 0$ in (13), that is, there are no inversions left after serving IC, then inversions would indeed have been converted to actual costs. Otherwise, suppose that after serving IC, there is an inversion between two items x and y, say, with x in front of y in the final offline list. Then by the definition of IC, item x was requested at least three more times after the last request to y. So the online player could have saved a cost unit by moving x in front of y in his list after the second request to x. To summarize, the sequence IC produces an additional online cost unit for every inversion that holds between the online and the offline list at the *end* of IC. Then, however, we can assume without loss of generality that both lists are in the same state after IC. Namely, if they are not, the online player could as well have served IC as intended (leaving no inversions) and invested the saved cost units in creating the inversions at the beginning of the next FLUP gadget, where their costs are taken into account. Thus, indeed, (13) holds with $E[I_{\text{after}}] = 0$ and inversions have become actual costs as stated in (11). The offline costs there are $v = OFF(IC) = 30$.

Since the online and offline list are identical at the end of the IC, a new FLUP game can be started. This generates request sequences of arbitrary length. In that way, we obtain a lower bound above 1.5 for the competitive ratio c in (2) for any additive constant b.

In the above construction, the value of the lower bound does not depend on whether the online player may use paid exchanges or not, but the adversary's strategy does use paid exchanges. So it seems that the online player cannot gain additional power from paid exchanges. This raises the conjecture that by restricting both players to free exchanges only, the list update problem might still have an optimal competitive ratio of 1.5. However, this is false. There is a randomized adversary strategy where the offline algorithm uses only free exchanges which cannot be served better than with a competitive ratio of $1.5+1/5048$. Because of the length of the sequences used in the corresponding FLUP game, this result is more difficult to obtain. First of all, the sequences used in that game are not found by brute force any more, but by dynamic game tree search with alpha-beta pruning in an approximate game. In that approximate game, the online player is restricted to a small set of random moves, similar to the poset algorithm. Secondly, the above argument about the order of the elements in the online list after leaving the IC gadget no longer holds. This can be resolved by a further elaboration of our method. The details are beyond the scope of this paper.

Extending our result to the full cost model requires a systematic treatment of lists of arbitrary length n. This is easy for the IC gadget but obviously difficult for the FLUP gadget. We hope to clarify the connection of FLUP with the sequences in (6) that beat the partial order approach to make progress in this direction.

References

[1] S. Albers (1995), Improved randomized on-line algorithms for the list update problem. *Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, 412–419.

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

[3] S. Albers, B. von Stengel, and R. Werchner (1996), List update posets. Manuscript.

[4] S. Albers and J. Westbrook (1998), A survey of self-organizing data structures. To appear in *Algorithmica.*

[5] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson (1994), On the power of randomization in on-line algorithms. *Algorithmica* 11, 2–14. Preliminary version in *Proc. 22nd STOC* (1990), 379–386.

[6] J. L. Bentley and C. C. McGeoch (1985), Amortized analyses of self-organizing sequential search heuristics. *Comm. ACM* 28, 404–411.

[7] A. Borodin and R. El-Yaniv (1998), Online Computation and Competitive Analysis. Cambridge Univ. Press, Cambridge.

[8] A. Borodin, N. Linial, and M. E. Saks (1992), An optimal online algorithm for metrical task systems. *J. ACM* 39, 745–763. Preliminary version in *Proc. 19th STOC* (1987), 373–382.

[9] S. Irani (1991), Two results on the list update problem, *Information Processing Letters* 38, 301–306.

[10] R. Karp and P. Raghavan (1990), unpublished.

[11] D. Koller, N. Megiddo, and B. von Stengel (1994), Fast algorithms for finding randomized strategies in game trees. *Proc. 26th STOC*, 750–759.

[12] H. W. Kuhn (1953), Extensive games and the problem of information. In: Contributions to the Theory of Games II, eds. H. W. Kuhn and A. W. Tucker, *Annals of Mathematics Studies* 28, Princeton Univ. Press, Princeton, 193–216.

[13] N. Reingold and J. Westbrook (1997), Optimum off-line algorithms for the list update problem. Technical Report YALEU/DCS/TR-805, Yale University.

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

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

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

[17] B. von Stengel (1996), Efficient computation of behavior strategies. *Games and Economic Behavior* 14, 220–246.

[18] A. C. Yao (1977), Probabilistic computations: Towards a unified measure of complexity, *Proc. 18th FOCS*, 222–227.