Improved Randomized On-Line Algorithms for the List Update Problem

Susanne Albers*

Abstract

The best randomized on-line algorithms known so far for the list update problem achieve a competitiveness of $\sqrt{3} \approx 1.73$. In this paper we present a new family of randomized on-line algorithms that beat this competitive ratio. Our improved algorithms are called TIMESTAMP algorithms and achieve a competitiveness of $\max\{2-p, 1+p(2-p)\}$, for any real number $p \in [0,1]$. Setting $p = (3-\sqrt{5})/2$, we obtain a ϕ-competitive algorithm, where $\phi = (1+\sqrt{5})/2 \approx 1.62$ is the Golden Ratio. TIMESTAMP algorithms coordinate the movements of items using some information on past requests. We can reduce the required information at the expense of increasing the competitive ratio. We present a very simple version of the TIMESTAMP algorithms that is 1.68-competitive. The family of TIMESTAMP algorithms also includes a new deterministic 2-competitive on-line algorithm that is different from the MOVE-TO-FRONT rule.

Keywords: Linear Lists, On-Line Algorithms, Competitive Analysis.

1 Introduction

The *list update problem* is among the first on-line problems that have been studied with respect to competitiveness. The problem consists in maintaining a set of items as an unsorted linear list. A list of n items is given. A list update algorithm is presented with a sequence of *requests* that must be served in their order of occurrence. Each request specifies an item in the list. In order to serve a request, a list update algorithm must *access* the requested item, i.e., it has to start at the front of the list and search linearly through the items until the desired item is found. Accessing the i-th item in the list incurs a cost of i. Immediately after an access, the requested item may be moved at no extra cost to any position closer to the front of the list. These exchanges are called *free exchanges*. All other exchanges of two consecutive items in the list cost 1 and are called *paid exchanges*. The goal is to serve the request sequence so that the total cost is as small as possible. A list update algorithm is *on-line* if it serves every request without knowledge of any future requests.

We analyze the performance of on-line algorithms for the list update problem using *competitive analysis* [6]. In a competitive analysis, an on-line algorithm A is compared to an *optimal off-line* algorithm. An optimal off-line algorithm knows the entire request sequence in advance and can serve it with minimum cost. Given a request sequence σ, let $C_A(\sigma)$ denote the cost incurred by

*Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. E-mail: albers@mpi-sb.mpg.de.
Work was supported in part by the ESPRIT Basic Research Actions Program of the EU under contract No. 7141 (project ALCOM II).

on-line algorithm A in serving σ and let $C_{OPT}(\sigma)$ denote the cost incurred by the optimal off-line algorithm OPT in processing σ. Then the algorithm A is called c-competitive if there is a constant a such that $C_A(\sigma) \leq c \cdot C_{OPT}(\sigma) + a$ for all request sequences σ. The *competitive factor* of A is the infimum of all c such that A is c-competitive.

Sleator and Tarjan [6] have shown that the well-known MOVE-TO-FRONT algorithm is 2-competitive. This deterministic on-line algorithm moves an item to the front of the list each time it is requested. Karp and Raghavan [4] have observed that no deterministic on-line algorithm for the list update problem can be better than 2-competitive. Thus the MOVE-TO-FRONT algorithm achieves the best possible competitive factor. A natural question is if the competitive factor of 2 can be improved using randomization. It was shown that against *adaptive adversaries*, no randomized on-line algorithm for the list update problem can be better than 2-competitive [1, 5]. Adaptive adversaries may see the on-line algorithm's random choices on past requests when generating a new request in a request sequence σ. On the other hand, against *oblivious adversaries*, the optimal competitive factor of randomized on-line algorithms has not been determined yet. An oblivious adversary specifies a request sequence in advance and is not allowed to see the random choices made by the on-line algorithm. A randomized on-line algorithm A is called c-competitive against any oblivious adversary if there exists a constant a such that for all request sequences σ generated by oblivious adversaries, $E[C_A(\sigma)] \leq c \cdot C_{OPT}(\sigma) + a$, where the expectation is taken over the random choices made by A. In this paper we always evaluate on-line algorithms with respect to oblivious adversaries. Irani [3] has exhibited the first randomized on-line algorithm for the list update problem; the SPLIT algorithm she proposed is $\frac{31}{16}$-competitive. Reingold *et al.* [5] have given a family of COUNTER and RANDOM RESET algorithms that achieve a competitive ratio of $\sqrt{3} \approx 1.73$. This has been the best upper bound known so far for randomized list update algorithms. The best lower bound known is due to Teia [7]. He shows that no randomized on-line algorithm for the list update problem can be better than 1.5-competitive.

In this paper we present improved randomized on-line algorithms for the list update problem that beat the competitive ratio of $\sqrt{3}$. Our new algorithms are called TIMESTAMP algorithms and achieve a competitiveness of $\max\{2-p, 1+p(2-p)\}$, for any real number $p \in [0,1]$. Choosing $p = (3-\sqrt{5})/2$, we obtain a ϕ-competitive algorithm, where $\phi = (1+\sqrt{5})/2 \approx 1.62$ is the Golden Ratio. TIMESTAMP algorithms move the requested item x not always to the front of the list but sometimes to a position that is only a bit closer to the front. This position can be computed easily when the algorithm scans the items preceding x in the list. However, in the implementation of the algorithm, the computation of this position requires a second pass through the list after the item x has been accessed. Moreover, some information on past requests is necessary in order to determine the desired position. We can simplify the TIMESTAMP algorithms so that they need less knowledge of previous requests; this increases the competitive ratio. We present a simplified version of the TIMESTAMP algorithms that is 1.68-competitive. The family of TIMESTAMP algorithms also includes two deterministic 2-competitive on-line algorithms, one of which is the MOVE-TO-FRONT rule. The second, new algorithm is the only other deterministic on-line algorithm found so far that achieves a competitive factor of 2; Sleator and Tarjan [6] have proved that the well-known deterministic algorithms TRANSPOSE and FREQUENCY COUNT are not 2-competitive.

The list update problem as defined above is the *static* version of the problem. Each request is an access to an item. In the *dynamic* variant of the problem, insertions and deletions of items are

allowed. A new item is inserted by scanning the entire list and appending the item at the end of the list. A deletion of an item is processed by searching for the item in the list and deleting it. In the following sections, when we develop and analyze randomized list update algorithms, we always consider the static version of the problem. However, the on-line algorithms we will propose can be extended in the obvious way so that they can handle insertions and deletions, too. All theorems that we will present also hold for the dynamic list update problem.

2 TIMESTAMP algorithms

We present a new family of randomized on-line algorithms for the list update problem. The following algorithm works for any real number $p \in [0,1]$.

Algorithm TIMESTAMP(p): Given a request sequence $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$, each request $\sigma(t)$, $1 \leq t \leq m$, is processed as follows. Suppose that $\sigma(t)$ is a request to item x.

With probability p, execute Step (a).

(a) Move x to the front of the list.

With probability $1-p$, execute Step (b).

(b) If x has not been requested so far during the time interval $[1, t-1]$, then do not change the position of x in the list. Otherwise let $t' \in [1, t-1]$ be the time at which x was requested most recently and serve the request $\sigma(t)$ as follows. Let $v_x(t)$ be the item closest to the front of the list that precedes x in the list and

(i) that was not requested during the interval $[t', t-1]$

or

(ii) that was requested exactly once during $[t', t-1]$ and the corresponding request was served using Step (b) of the algorithm.

If there is no such item, then let $v_x(t) = x$.

Insert x immediately before $v_x(t)$.

Theorem 1 *For any real number $p \in [0,1]$, TIMESTAMP(p) is c-competitive, where $c = \max\{2-p, 1+p(2-p)\}$.*

Corollary 1 *TIMESTAMP($\frac{3-\sqrt{5}}{2}$) is ϕ-competitive, where $\phi = \frac{1}{2}(1+\sqrt{5})$ is the Golden Ratio.*

An interesting feature of the TIMESTAMP(p) algorithm is that at a request to item x, all items y satisfying condition (i) or (ii) in Step (b) are stored consecutively in front of x. In other words, all items stored between x and $v_x(t)$ satisfy condition (i) or (ii); we will prove this later in Lemma 2.

We assume that the algorithm TIMESTAMP(p) maintains a time stamp $ST(y)$ for each item y in the list. While a request sequence in served, $ST(y)$ always stores the time of the most recent request to y. When there is a request to item x, TIMESTAMP(p) can easily determine the first item in the list that satisfies condition (i) in Step (b). The algorithm just has to find the first item y with $ST(y) < ST(x)$. Finding the first item in the list that satisfies condition (ii) in Step (b) requires some more information on previous requests. We can simplify the TIMESTAMP(p) algorithm at the expense of increasing the competitive ratio. Suppose that we drop condition (ii) in Step (b). Then $v_x(t)$ is simply the item closest to the front of the list that precedes x and has not been requested during the interval $[t', t-1]$. We can show the following performance.

Theorem 2 *If condition (ii) is dropped in Step (b) of TIMESTAMP(p), then the resulting algorithm is c-competitive, where* $c = \max\{2-p, 1+p(2-p), 2-\frac{3}{2}p+2p^2-\frac{1}{2}p^3\}$. *Setting* $p = \frac{1}{2}(5-\sqrt{17})$, *we obtain a competitive ratio of* $\frac{3}{2}(\sqrt{17}-3) \approx 1.68$.

We will prove this theorem after we have shown Theorem 1.

Although the item $v_x(t)$ specified in TIMESTAMP(p) can be computed easily, in a real implementation of the algorithm, we need a second pass through the list in order to actually locate $v_x(t)$. Note that a number of on-line algorithms that have been proposed in the literature, such as the algorithm FREQUENCY COUNT, also require such a second pass after each access to an item. There is an alternative formulation of TIMESTAMP(p) that does not need a second pass. In this alternative formulation, we maintain a pointer $ptr(x)$ for each item x in the list; $ptr(x)$ either points to x or to an item preceding x in the list. At a request to item x, x is inserted at the front of the list or immediately before $ptr(x)$. The element specified by $ptr(x)$ always corresponds to $v_x(t)$. The drawback of this alternative formulation is that we need time to update the pointers. Essentially, at a request to item x, the pointers of all elements y with $ptr(y) = x$ must be set to the successor of x in the list, and $ptr(x)$ must be set to the front of the list. We prefer the formulation of the TIMESTAMP(p) algorithm given above because it explicitly describes the properties of the position $v_x(t)$ in front of which an item x is to be inserted.

TIMESTAMP(p) describes two deterministic algorithms. Setting $p = 1$ we obtain the MOVE-TO-FRONT algorithm. Theorems 1 and 2 confirm the well-known fact that the MOVE-TO-FRONT rule is 2-competitive. On the other hand, assume $p = 0$ and consider the simplified version of the TIMESTAMP algorithm in which condition (ii) of Step (b) is dropped. The resulting deterministic algorithm always inserts the requested item x immediately before the first item in the list that has not been requested since the last request to x. Theorem 2 implies that this deterministic strategy is 2-competitive.

We now proceed with the proof of Theorem 1. Consider a fixed $p \in [0,1]$. Let $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$ be an arbitrary request sequence consisting of m requests and let $\sigma(t)$ denote the request at time t, $1 \leq t \leq m$. We first present two lemmata that describe the relative positions of items in the list while TIMESTAMP(p) serves a request sequence.

Lemma 1 *Let x and y, $x \neq y$, be two items. Suppose that x is requested at time t' and at time t, $t' < t$, and that y is not requested during the interval $[t', t]$. Then, immediately after the service of $\sigma(t)$, x precedes y in TIMESTAMP(p)'s list and this relation does not change before the next request to y.*

Proof: If TIMESTAMP(p) executes Step (a) when serving $\sigma(t)$, then x is moved to the front of the list and must precede y in the list. If TIMESTAMP(p) executes Step (b) when serving $\sigma(t)$, then condition (i) in Step (b) of the algorithm ensures that x is inserted at some position in front of item y because y is not requested during $[t', t]$. In any case x precedes y in TIMESTAMP(p)'s list immediately after the service of $\sigma(t)$. Since y is only moved when it is requested, the relative position of x and y cannot change before the next request to y. □

Lemma 2 *Let t be a time in $[1, m]$. If the item $x = \sigma(t)$ was requested at least once in $[1, t-1]$, then the following two statements hold. Let t', $t' < t$, denote the time at which x was requested most recently.*

a) If item y, $y \neq x$, was requested at least twice during $[t', t-1]$, then y precedes item $v_x(t)$ in TIMESTAMP(p)'s list at time t.

b) If item y, $y \neq x$, was requested exactly once during $[t', t-1]$ and the corresponding request was served using Step (a) of TIMESTAMP(p), then y precedes item $v_x(t)$ in TIMESTAMP(p)'s list at time t.

Proof: Suppose that there is a time in $[1, m]$ at which Lemma 2 does not hold. Then let $t_0 \in [1, m]$ be the earliest point of time at which the lemma is violated. Furthermore, let t_0', $t_0' < t_0$, be the time at which item $x = \sigma(t_0)$ was requested most recently, and let $z = v_x(t_0)$.

First we examine the case that statement a) of the lemma does not hold. Thus, there exists an item y, $y \neq x$, that is requested at least twice in $[t_0', t_0 - 1]$ and that does not precede z at time t_0. Let t_y be the time of the last request to y in $[t_0', t_0 - 1]$. We show that after the service of $\sigma(t_y)$, item y precedes z in TIMESTAMP(p)'s list. If $\sigma(t_y)$ is served using Step (a) of the algorithm, then there is nothing to show. Suppose that $\sigma(t_y)$ is served using Step (b) of TIMESTAMP(p). By the definition of $v_x(t_0)$, item z is requested at most once in $[t_0', t_0 - 1]$, and such a request is served using Step (b) of the algorithm. This implies that when TIMESTAMP(p) serves $\sigma(t_y)$, z cannot precede $v_y(t_y)$ (due to conditions (i) and (ii) in Step (b) of the algorithm). Hence y is inserted at some position in front of z. We conclude that y must precede z after the service of $\sigma(t_y)$. Since, by assumption, z precedes y at time t_0, item z must be requested at some time $t_z \in [t_y + 1, t_0 - 1]$ and z must be inserted at some position in front of y when $\sigma(t_z)$ is served. This implies that y cannot precede $v_z(t_z)$ at time t_z because $\sigma(t_z)$ is served using Step (b) of the algorithm. Note that at time t_z, y was requested at least twice since the last request to z. Hence statement a) of the lemma does not hold at time t_z, and we have a contradiction to the minimality of t_0.

Now assume that statement b) of the lemma is violated. Let y, $y \neq x$, be an item for which statement b) does not hold, and let $t_y \in [t_0', t_0 - 1]$ be the time at which y is requested. By assumption, y does not precede $z = v_x(t_0)$ at time t_0. Since $\sigma(t_y)$ is served using Step (a) of the algorithm, y precedes z after the service of $\sigma(t_y)$. Using the same arguments as above, we can derive a contradiction to the choice of t_0. □

In the following we will evaluate TIMESTAMP(p)'s and OPT's cost on request sequence σ. Let $C_{TS}(\sigma)$ be the cost incurred by TIMESTAMP(p) in serving σ. We will show that

$$E[C_{TS}(\sigma)] \leq c \cdot C_{OPT}(\sigma), \tag{1}$$

where $c = \max\{2 - p, 1 + p(2 - p)\}$. This proves Theorem 1. Here we assume without loss of generality that TIMESTAMP(p) and OPT start with the same initial list. In the following, when analyzing on-line and off-line cost, we will always use the $(i-1)$-*cost measure*, i.e., we assume that an access to the i-th item in the list incurs a cost of $i - 1$ rather than i. Obviously, an on-line algorithm that is c-competitive in the $(i-1)$-cost measure is also c-competitive in the i-cost measure.

We need some notation. Let L be the set of items in the list. For any $t \in [1, m]$ and any item $x \in L$, let $C_{TS}(t, x)$ be the cost incurred by item x when TIMESTAMP(p) serves $\sigma(t)$. More precisely, $C_{TS}(t, x) = 1$ if at time t, item x precedes the item requested by $\sigma(t)$ in TIMESTAMP(p)'s

list; otherwise $C_{TS}(t,x) = 0$. We have

$$\begin{aligned} E[C_{TS}(\sigma)] &= E[\sum_{t\in[1,m]}\sum_{x\in L} C_{TS}(t,x)] \\ &= E[\sum_{x\in L}\sum_{t\in[1,m]} C_{TS}(t,x)] \\ &= E[\sum_{x\in L}\sum_{y\in L}\sum_{\substack{t\in[1,m]\\ \sigma(t)=y}} C_{TS}(t,x)]. \end{aligned}$$

Let $\{x,y\}$ be an unordered pair of items x and y with $x \neq y$. Every pair $\{x,y\}$ contributes two terms in the last line of the above equation, namely

$$\sum_{\substack{t\in[1,m]\\ \sigma(t)=x}} C_{TS}(t,y) \quad \text{and} \quad \sum_{\substack{t\in[1,m]\\ \sigma(t)=y}} C_{TS}(t,x).$$

Thus,

$$E[C_{TS}(\sigma)] = E[\sum_{\substack{\{x,y\}\\ x\neq y}} (\sum_{\substack{t\in[1,m]\\ \sigma(t)=x}} C_{TS}(t,y) + \sum_{\substack{t\in[1,m]\\ \sigma(t)=y}} C_{TS}(t,x))]. \tag{2}$$

The cost incurred by OPT can be written in a similar way, i.e.,

$$C_{OPT}(\sigma) = \sum_{\substack{\{x,y\}\\ x\neq y}} (\sum_{\substack{t\in[1,m]\\ \sigma(t)=x}} C_{OPT}(t,y) + \sum_{\substack{t\in[1,m]\\ \sigma(t)=y}} C_{OPT}(t,x) \; + \; p(x,y)). \tag{3}$$

Here $C_{OPT}(t,y)$ and $C_{OPT}(t,x)$ denote the costs incurred by items y and x when OPT serves $\sigma(t)$. For any unordered pair $\{x,y\}$ of items $x \neq y$, $p(x,y)$ denotes the total number of paid exchanges that OPT incurs in moving x in front of y or y in front of x.

Now, for any pair $\{x,y\}$ of items with $x \neq y$, let σ_{xy} be the request sequence that is obtained from σ if we delete all requests in σ that are neither to x nor to y. Let $E[C_{TS}(\sigma_{xy})]$ be the expected cost incurred by TIMESTAMP(p) if it serves σ_{xy} on a list consisting of x and y only. Lemma 2 implies that if TIMESTAMP(p) serves a request $\sigma(t)$ in σ using Step (b), then the requested item $x = \sigma(t)$ never passes an item that was requested at least twice since the last request to x or that was requested exactly once and the corresponding request was served using Step (a) of the algorithm. Thus, for any pair $\{x,y\}$ of items, the following statement holds. If TIMESTAMP(p) serves σ on the entire list, then the relative position of x and y changes in the same way as if TIMESTAMP(p) is run of the two item list consisting of x and y with request sequence σ_{xy}. Therefore, we have

$$E[C_{TS}(\sigma_{xy})] = E[\sum_{\substack{t\in[1,m]\\ \sigma(t)=x}} C_{TS}(t,y) + \sum_{\substack{t\in[1,m]\\ \sigma(t)=y}} C_{TS}(t,x)]$$

and, by equation (2),

$$E[C_{TS}(\sigma)] = \sum_{\substack{\{x,y\}\\ x\neq y}} E[C_{TS}(\sigma_{xy})].$$

Let $C_{OPT}(\sigma_{xy})$ be the cost incurred by OPT if it serves σ_{xy} on the list consisting of x and y only. In this two item list, OPT can always arrange x and y optimally, which might not be possible when

OPT serves σ on the entire list. Hence

$$C_{OPT}(\sigma_{xy}) \leq \sum_{\substack{t\in[1,m]\\ \sigma(t)=x}} C_{OPT}(t,y) + \sum_{\substack{t\in[1,m]\\ \sigma(t)=y}} C_{OPT}(t,x) \ + \ p(x,y)$$

and equation (3) implies

$$C_{OPT}(\sigma) \geq \sum_{\substack{\{x,y\}\\ x\neq y}} C_{OPT}(\sigma_{xy}).$$

This method of analyzing cost by considering pairs of items was also used in [2, 3]. In the following we show that for any pair of items $\{x, y\}$ with $x \neq y$,

$$E[C_{TS}(\sigma_{xy})] \leq c \cdot C_{OPT}(\sigma_{xy}), \tag{4}$$

where $c = \max\{2-p, 1+p(2-p)\}$. This proves inequality (1).

Consider an arbitrary but fixed pair $\{x, y\}$ with $x \neq y$. Let $\sigma_{xy} = \sigma(t_1), \sigma(t_2), \ldots, \sigma(t_k)$ for some non-negative integer k. For $i = 1, 2, \ldots, k$, let u_i be the item requested by $\sigma(t_i)$. Lemma 1 suggests to partition σ_{xy} into phases $P(1), P(2), \ldots, P(l)$ for some l so that the following condition holds. If phase $P(j)$, $1 \leq j \leq l$, starts at time t_{b_j}, then it ends at time t_{e_j}, where

$$e_j = \min\{i > b_j | u_{i-1} = u_i \text{ and } u_i \neq u_{i+1}\}.$$

In words, a phase ends when, for the first time, there have been two consecutive requests to the same item and the next request is different. The phases we obtain can be classified as follows.

Type 1:	(a) $P(j) = x^h$	or	(b) $P(j) = y^h$	for some $h \geq 2$
Type 2:	(a) $P(j) = xy^h$	or	(b) $P(j) = yx^h$	for some $h \geq 2$
Type 3:	(a) $P(j) = (xy)^{h_1}x^{h_2}$	or	(b) $P(j) = (yx)^{h_1}y^{h_2}$	for some $h_1 \geq 1, h_2 \geq 2$
Type 4:	(a) $P(j) = (xy)^{h_1}y^{h_2}$	or	(b) $P(j) = (yx)^{h_1}x^{h_2}$	for some $h_1 \geq 2, h_2 \geq 1$

We may assume without loss of generality that the item first requested in a phase $P(j)$, $1 \leq j \leq l$, is behind the other item of the pair $\{x, y\}$ in the two item lists maintained by TIMESTAMP(p) and OPT. This is easy to see for phase $P(1)$. If the first item in $P(1)$, say x, precedes y in the initial list, then we can simply omit the first requests to x in σ_{xy} until we obtain the first request to y. This does not change the cost incurred by TIMESTAMP(p) and OPT in $P(1)$ because we assume that TIMESTAMP(p) and OPT start with the same initial list. Now consider a phase $P(j)$, $2 \leq j \leq l$. The item first requested in $P(j)$ differs from the item requested by the two previous requests. Lemma 1 immediately implies that the first item in $P(j)$ is behind the other item of the pair $\{x, y\}$ in the list maintained by TIMESTAMP(p). Consider OPT's movements when it serves the request sequence σ_{xy} on the two item list. We may assume without loss of generality that whenever there are two consecutive requests to the same item, OPT moves that item to the front of the list, provided that it has not been there yet. This implies without loss of generality that immediately before the first request in a phase $P(j)$, the item requested first in the phase is also behind the other item of the pair $\{x, y\}$ in OPT's two item list.

In the following we evaluate the expected cost incurred by TIMESTAMP(p) and the cost incurred by OPT in each phase of σ_{xy}. For each $j = 1, 2, \ldots, l$, the expected cost incurred by

TIMESTAMP(p) in phase $P(j)$ of σ_{xy} is

$$E[C_{TS}(P(j))] = E[\sum_{i=b_j}^{e_j}(C_{TS}(t_i, y) + C_{TS}(t_i, x))].$$

Similarly, for $j = 1, 2, \ldots, l$, let $C_{OPT}(P(j))$ be the cost incurred by OPT when it serves phase $P(j)$ of σ_{xy}. We will prove the following lemmata.

Lemma 3 *If $P(j)$ has type 1, then $E[C_{TS}(P(j))] \leq (2-p)C_{OPT}(P(j))$.*

Lemma 4 *If $P(j)$ has type 2, then $E[C_{TS}(P(j))] \leq (1+p(2-p))C_{OPT}(P(j))$.*

Lemma 5 *If $P(j)$ has type 3 or type 4, then $E[C_{TS}(P(j))] \leq \max\{2-p, 1+p(2-p)\}C_{OPT}(P(j))$.*

Before we prove the lemmata we finish the proof of inequality (4). Lemmata 3 – 5 imply that

$$\begin{aligned} E[C_{TS}(\sigma_{xy})] &= E[\sum_{j=1}^{l} C_{TS}(P(j))] \\ &\leq \max\{2-p, 1+p(2-p)\}\sum_{j=1}^{l} C_{OPT}(P(j)) \\ &= \max\{2-p, 1+p(2-p)\}C_{OPT}(\sigma_{xy}) \end{aligned}$$

and inequality (4) is proved.

We have classified phases $P(1), P(2), \ldots, P(l)$ into four types. For each type, subtypes (a) and (b) are symmetric to each other. In the following, we will always assume without loss of generality that the considered phase has subtype (a). Let $P(j)$ be an arbitrary phase. By the above discussion we know that immediately before the first request in $P(j)$, x is behind y in the two item lists maintained by TIMESTAMP(p) and OPT. Thus $E[C_{TS}(t_{b_j}, y)] = C_{OPT}(t_{b_j}, y) = 1$, for $j = 1, 2, \ldots, l$.

Claim 1 below will be useful when proving Lemmata 3 – 5. We will present a proof of this claim later.

Claim 1 *After the service of the first request $\sigma(t_{b_j})$ in a phase $P(j)$, $1 \leq j \leq l$, item $x = \sigma(t_{b_j})$ precedes item y if and only if TIMESTAMP(p) serves $\sigma(t_{b_j})$ using Step (a).*

Proof of Lemma 3: We have $C_{TS}(P(j)) = \sum_{i=b_j}^{e_j} C_{TS}(t_i, y)$. By Lemma 1, x precedes y in TIMESTAMP(p)'s list after the service of $\sigma(t_{b_j+1})$. Hence, y cannot cause a cost at the third and all remaining requests to x in $P(j)$. We obtain

$$\begin{aligned} E[C_{TS}(P(j))] &= E[C_{TS}(t_{b_j}, y)] + E[C_{TS}(t_{b_j+1}, y)] \\ &= 1 + E[C_{TS}(t_{b_j+1}, y)]. \end{aligned}$$

$E[C_{TS}(t_{b_j+1}, y)]$ is the probability that item $x = u_{b_j+1}$ is behind y in TIMESTAMP(p)'s list when $\sigma(t_{b_j+1})$ is served. Applying Claim 1 we infer that x is behind y if and only if TIMESTAMP(p) serves

$\sigma(t_{b_j})$ using Step (b), which happens with probability $1-p$. We conclude $E[C_{TS}(t_{b_j+1}, y)] = 1-p$ and $E[C_{TS}(P(j))] = 2-p$. Since $C_{OPT}(P(j)) = 1$, the lemma follows. □

Proof of Lemma 4: TIMESTAMP(p)'s cost in phase $P(j)$ is $C_{TS}(P(j)) = C_{TS}(t_{b_j}, y) + \sum_{i=b_j+1}^{e_j} C_{TS}(t_i, x)$. Lemma 1 implies that x cannot cause a cost at the third and all remaining requests to y in $P(j)$. Hence

$$\begin{aligned} E[C_{TS}(P(j))] &= E[C_{TS}(t_{b_j}, y)] + E[C_{TS}(t_{b_j+1}, x)] + E[C_{TS}(t_{b_j+2}, x)] \\ &= 1 + E[C_{TS}(t_{b_j+1}, x)] + E[C_{TS}(t_{b_j+2}, x)]. \end{aligned}$$

We have $C_{TS}(t_{b_j+1}, x) = 1$ if the TIMESTAMP(p) algorithm moves x in front of y when serving $\sigma(t_{b_j})$. By Claim 1, this happens if TIMESTAMP(p) processes $\sigma(t_{b_j})$ using Step (a). Therefore, $E[C_{TS}(t_{b_j+1}, x)] = p$. We analyze $E[C_{TS}(t_{b_j+2}, x)]$. We have $C_{TS}(t_{b_j+2}, x) = 1$ if TIMESTAMP(p) moves x in front of y when serving $\sigma(t_{b_j})$ and does not move y in front of x when serving $\sigma(t_{b_j+1})$. Claim 1 implies that with probability p, TIMESTAMP(p) moves x in front of y when serving $\sigma(t_{b_j})$. Item y can only stay behind x in TIMESTAMP(p)'s list if $\sigma(t_{b_j+1})$ is served using Step (b), which happens with probability $1-p$. Thus, the expected cost on $\sigma(t_{b_j+2})$ is at most $p(1-p)$. We obtain $E[C_{TS}(P(j))] \leq 1 + p + p(1-p) = 1 + p(2-p)$, and the lemma follows because $C_{OPT}(P(j)) = 1$. □

Proof of Lemma 5: We know $C_{TS}(t_{b_j}, y) = 1$ and, using Claim 1, $E[C_{TS}(t_{b_j+1}, x)] = p$. First we assume that $P(j)$ has type 3. Then $P(j)$ consists of a head of $2h_1$ alternating requests to x and y and of a tail of h_2 requests to x. Lemma 1 ensures that in the tail of requests to x, item x must precede y in TIMESTAMP(p)'s list after the service of the second request to x. Hence y cannot cause a cost at any of the remaining requests in $P(j)$, i.e., $C_{TS}(t_i, y) = 0$ for $i = b_j + 2(h_1+1), \ldots, e_j$. Thus

$$C_{TS}(P(j)) = 1 + p + \sum_{i=b_j+2}^{b_j+2h_1} C_{TS}(t_i, u_{i-1}) + C_{TS}(t_{b_j+2h_1+1}, y).$$

For the analysis of the cost incurred at the alternating requests to x and y we need the following claim that we will prove later.

Claim 2 *Let $\sigma(t_{i-3})\sigma(t_{i-2})\sigma(t_{i-1}) = xyx$ or $\sigma(t_{i-3})\sigma(t_{i-2})\sigma(t_{i-1}) = yxy$ be three consecutive requests in σ_{xy} with $4 \leq i \leq k$. Then, after the service of $\sigma(t_{i-1})$, with probability $1-p+p^2$, u_{i-1} precedes u_{i-2} in the list maintained by TIMESTAMP(p).*

If we have a phase $P(j)$ with $j \geq 2$, then Claim 2 implies that, for $i = b_j + 2, \ldots, b_j + 2h_1$, immediately before the request to $\sigma(t_i)$, item u_{i-1} precedes $u_i = u_{i-2}$ with probability $1-p+p^2$. Therefore, $E[C_{TS}(t_i, u_{i-1})] = 1-p+p^2$ for $i = b_j + 2, \ldots, b_j + 2h_1$. We remark that we may apply Claim 2 for $i = b_j + 2$ because $u_{b_j-1} \neq u_{b_j}$. For phase $P(1)$, Claim 2 gives $E[C_{TS}(t_i, u_{i-1})] = 1-p+p^2$ for $i = b_1 + 3, \ldots, b_1 + 2h_1$. However, it is easy to show that for request $\sigma(t_{b_1+2})$, $E[C_{TS}(t_{b_1+2}, u_{b_1+1})] = 1-p+p^2$. We now evaluate the cost $C_{TS}(t_{b_j+2h_1+1}, y)$. Claim 2 gives that immediately before the request $\sigma(t_{b_j+2h_1+1})$, item x precedes item y with probability $1-p+p^2$ in TIMESTAMP(p)'s list. Hence $E[C_{TS}(t_{b_j+2h_1+1}, y)] = 1-(1-p+p^2) = p-p^2$. We conclude

$$\begin{aligned} E[C_{TS}(P(j))] &= 1 + p + (2h_1 - 1)(1-p+p^2) + p - p^2 \\ &= 2(h_1+1)(1-p+p^2) - 2 + 5p - 4p^2 \\ &\leq 2(h_1+1)(1-p+p^2) \end{aligned}$$

for all $p \in [0,1]$. Thus

$$\begin{aligned} E[C_{TS}(P(j))] &\leq 2(h_1+1)(1-p+p^2) \\ &= (h_1+1)(2-p)+(h_1+1)(-p+2p^2) \\ &\leq (h_1+1)(2-p) \end{aligned}$$

for all $p \leq \frac{1}{2}$. Moreover

$$\begin{aligned} E[C_{TS}(P(j))] &\leq 2(h_1+1)(1-p+p^2) \\ &= (h_1+1)(1+p(2-p))+(h_1+1)(1-4p+3p^2) \\ &\leq (h_1+1)(1+p(2-p)) \end{aligned}$$

for all $p \geq \frac{1}{3}$. Since $C_{OPT}(P(j)) = h_1 + 1$, we obtain

$$E[C_{TS}(P(j))] \leq \max\{2-p, 1+p(2-p)\} C_{OPT}(P(j)).$$

The analysis for a phase $P(j)$ having type 4 is very similar. By Lemma 1, $C_{TS}(t_i, x) = 0$ for $i = b_j + 2h_1 + 1, \ldots, e_j$. Hence

$$C_{TS}(P(j)) = 1 + p + \sum_{i=b_j+2}^{b_j+2h_1-1} C_{TS}(t_i, u_{i-1}) + C_{TS}(t_{b_j+2h_1}, x).$$

Applying Claim 2 we obtain $E[C_{TS}(t_i, u_{i-1})] = 1 - p + p^2$, for $i = b_j + 2, \ldots, b_j + 2h_1 - 1$, and $E[C_{TS}(t_{b_j+2h_1}, x)] = p - p^2$. Thus

$$\begin{aligned} E[C_{TS}(P(j))] &= 1 + p + 2(h_1-1)(1-p+p^2) + p - p^2 \\ &= h_1(2-p) + h_1(-p+2p^2) - 1 + 4p - 3p^2 \\ &\leq h_1(2-p) \end{aligned}$$

for all $p \leq \frac{1}{3}$. Furthermore we can show

$$\begin{aligned} E[C_{TS}(P(j))] &= h_1(1+p(2-p)) + (h_1-1)(1-4p+3p^2) \\ &\leq h_1(1+p(2-p)) \end{aligned}$$

for all $p \geq \frac{1}{3}$. We have $C_{OPT}(P(j)) = h_1$ and therefore

$$E[C_{TS}(P(j))] \leq \max\{2-p, 1+p(2-p)\} C_{OPT}(P(j)). \quad \square$$

Proof of Claim 1: We know that item x is behind y in TIMESTAMP(p)'s list before the service of $\sigma(t_{b_j})$. If TIMESTAMP(p) executes Step (a) when serving $\sigma(t_{b_j})$, then x is moved to the front of the list and must precede y. On the other hand, suppose that $\sigma(t_{b_j})$ is served using Step (b). If x was not requested during $[1, t_{b_j} - 1]$, then the position of x remains unchanged and x stays behind y in the list. If x was requested at least once in $[1, t_{b_j} - 1]$, then Lemma 2a) implies that y precedes $v_x(t_{b_j})$ at time t_{b_j}. Again, x cannot be moved in front of y during the service of $\sigma(t_{b_j})$. $\square$

Proof of Claim 2: We analyze the sequence $\sigma(t_{i-3})\sigma(t_{i-2})\sigma(t_{i-1}) = xyx$. The case $\sigma(t_{i-3})\sigma(t_{i-2})\sigma(t_{i-1}) = yxy$ is symmetric. We have to compute the probability that x precedes

y in TIMESTAMP(p)'s list after the service of $\sigma(t_{i-1})$. If TIMESTAMP(p) serves $\sigma(t_{i-1})$ using Step (a) of the algorithm, then x is moved to the front of the list and precedes y. Now assume that $\sigma(t_{i-1})$ is served using Step (b) of TIMESTAMP(p). If $\sigma(t_{i-2})$ was processed using Step (b) of the algorithm, then condition (ii) in Step (b) ensures that x is inserted at some position in front of y when TIMESTAMP(p) serves $\sigma(t_{i-1})$. On the other hand, if $\sigma(t_{i-2})$ was processed using Step (a) of the algorithm, then Lemma 2b) implies that y precedes $v_x(t_{i-1})$ and x is inserted behind y when TIMESTAMP(p) serves $\sigma(t_{i-1})$.

We conclude that item x precedes item y in TIMESTAMP(p)'s list after the service of $\sigma(t_{i-1})$ if and only if one of the following events occurs. (A) TIMESTAMP(p) serves $\sigma(t_{i-1})$ using Step (a); (B) TIMESTAMP(p) serves $\sigma(t_{i-2})$ and $\sigma(t_{i-1})$ using Step (b). Event A occurs with probability p whereas event B occurs with probability $(1-p)^2$. Thus, with probability $p+(1-p)^2 = 1-p+p^2$, x precedes y in TIMESTAMP(p)'s list after the service of $\sigma(t_{i-1})$. □

In the above analysis we assume that the last phase $P(l)$ is a full phase of one of the phase types 1 – 4. It is easy to see that Lemmata 3 – 5 also hold for a phase $P(l)$ that is a prefix of one of the phase types. The proof of Theorem 1 is complete.

Next we analyze the performance of the simplified TIMESTAMP(p) algorithm.

Proof of Theorem 2: The proof has the same structure as the proof of Theorem 1. The statements of Lemma 1 and Lemma 2 can be shown in the same way as before. The proofs of the lemmata become simpler, though, because we do not have to consider items that satisfy condition (ii) in Step (b) of the TIMESTAMP(p) algorithm. We compare the on-line and off-line cost for each pair $\{x, y\}$ of items with $x \neq y$ and prove

$$E[\sum_{\substack{t\in[1,m]\\ \sigma(t)=x}} C_{TS}(t,y) + \sum_{\substack{t\in[1,m]\\ \sigma(t)=y}} C_{TS}(t,x)] \leq c \cdot C_{OPT}(\sigma_{xy}), \tag{5}$$

where $c = \max\{2-p, 1+p(2-p), 2-\frac{3}{2}p+p^2-\frac{1}{2}p^3\}$. We partition the request sequence σ_{xy} in the same way as before. Lemmata 3 and 4 as well as their proofs (including Claim 1) remain the same. In the proof of Claim 2, only a weaker statement can be shown. We know that item u_{i-1} precedes item u_{i-2} after the service of $\sigma(t_{i-1})$ if the simplified TIMESTAMP(p) algorithm serves $\sigma(t_{i-1})$ using Step (a). Also, u_{i-1} cannot precede u_{i-2} if the simplified algorithm serves $\sigma(t_{i-2})$ using Step (a) and $\sigma(t_{i-1})$ using Step (b). Unfortunately, we do not have information about the relative position of u_{i-2} and u_{i-1} if the simplified algorithm processes $\sigma(t_{i-2})$ and $\sigma(t_{i-1})$ using Step (b). Therefore, Claim 2 changes to the following statement.

Claim 3 *Let $\sigma(t_{i-3})\sigma(t_{i-2})\sigma(t_{i-1}) = xyx$ or $\sigma(t_{i-3})\sigma(t_{i-2})\sigma(t_{i-1}) = yxy$ be three consecutive requests in σ_{xy} with $4 \leq i \leq k$. Then, after the service of $\sigma(t_{i-1})$, with probability at most $1-p+p^2$, u_{i-1} precedes u_{i-2} in the list maintained by the simplified TIMESTAMP(p) algorithm.*

For a proof of a statement corresponding to Lemma 5, we first consider a phase $P(j)$ having type 3. Again

$$C_{TS}(P(j)) \;=\; 1+p+\sum_{i=b_j+2}^{b_j+2h_1} C_{TS}(t_i, u_{i-1}) + C_{TS}(t_{b_j+2h_1+1}, y).$$

We examine the cost $C_{TS}(t_{b_j+2h_1+1}, y)$. We have $C_{TS}(t_{b_j+2h_1+1}, y) = 1$ if $C_{TS}(t_{b_j+2h_1}, y) = 1$ and item $x = u_{b_j+2h_1}$ is not moved in front of y at time $t_{b_j+2h_1}$. Item x can only stay behind y if $\sigma(t_{b_j+2h_1})$ is served using Step (b) of the algorithm. The event that the simplified TIMESTAMP algorithm serves $\sigma(t_{b_j+2h_1})$ using Step (b) is independent of the event that $C_{TS}(t_{b_j+2h_1}, y) = 1$. Hence $E[C_{TS}(t_{b_j+2h_1+1}, y)] \leq (1-p+p^2)(1-p)$. This implies

$$E[C_{TS}(P(j))] \leq 1 + p + (2h_1 - 1)(1 - p + p^2) + (1-p)(1-p+p^2).$$

Using the same techniques as in the proof of Lemma 5 we can show

$$E[C_{TS}(P(j))] \leq \max\{2-p, 1+p(2-p)\} \cdot C_{OPT}(P(j)).$$

Now consider a phase $P(j)$ having type 4. We have

$$E[C_{TS}(P(j))] \leq 1 + p + 2(h_1 - 1)(1 - p + p^2) + (1-p)(1-p+p^2).$$

Hence

$$\begin{aligned} E[C_{TS}(P(j))] &\leq 2h_1(1-p+p^2) - (1+p)(-p+p^2) \\ &= h_1(2-2p+2p^2) + p(1-p^2). \end{aligned}$$

Since $h_1 \geq 2$,

$$\begin{aligned} E[C_{TS}(P(j))] &\leq h_1(2-2p+2p^2 + \frac{p}{2}(1-p^2)) \\ &= h_1(2 - \frac{3}{2}p + 2p^2 - \frac{1}{2}p^3). \end{aligned}$$

The statement corresponding to Lemma 5 is

Lemma 6 *If $P(j)$ has type 3 or type 4, then $E[C_{TS}(P(j))] \leq c \cdot C_{OPT}(P(j))$, where $c = \max\{2-p, 1+p(2-p), 2 - \frac{3}{2}p + 2p^2 - \frac{1}{2}p^3\}$.*

Lemma 6 and the statements of Lemmata 3 and 4 imply inequality (5). The proof of Theorem 2 is complete. □

We conclude with some remarks. TIMESTAMP algorithms use $\Theta(m)$ random bits on a request sequence of length m. We can modify the original version of TIMESTAMP(p) so that it uses only $O(n)$ random bits during an initialization phase and runs completely deterministically thereafter. The competitive ratio is still $c = \max\{2-p, 1+p(2-p)\}$. The idea is to have two different types of items, item type (a) and item type (b). Initially, we decide for each item in the list which type it should have. With probability p an item has type (a), and with probability $1-p$ it has type (b); the initializations are done independently. When a request sequence is served, each request to a type (a) item is served using Step (a) of the algorithm and every request to a type (b) item is served using Step (b). This technique cannot be applied in the simplified TIMESTAMP algorithm in which condition (ii) in Step (b) is dropped. Our analysis of the simplified TIMESTAMP algorithm makes use of the fact that the decision whether a given request is processed using Step (a) or (b) does not depend on previous requests (see the analysis after Claim 3). In the simplified TIMESTAMP algorithm we can reduce the number of random bits using a technique presented by Reingold *et*

al. [5]. For each item in the list we maintain a mod i counter, where i is a positive integer. These counters are initialized independently and uniformly at random to a value in $\{0, 1, \ldots, i-1\}$. Furthermore, we choose a non-empty subset I of $\{0, 1, \ldots, i-1\}$. At a request to item x, the TIMESTAMP algorithm first decrements x's counter by 1. If the counter is I, the algorithm serves the request using Step (a), otherwise it executes Step (b). Choosing i and I appropriately, we can achieve a competitive ratio of $c+\epsilon$ for any $\epsilon > 0$; here $c = \max\{2-p, 1+p(2-p), 2-\frac{3}{2}p+2p^2-\frac{1}{2}p^3\}$.

Acknowledgement

The author thanks Rudolf Fleischer and Stefan Schirra for reading an earlier version of this manuscript.

References

[1] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, **11**:2–14, 1994.

[2] J.L. Bentley and C.C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. *Communication of the ACM*, **28**:404–411, 1985.

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

[4] R. Karp and P. Raghavan. From a personal communication cited in [5].

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

[6] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communication of the ACM*, **28**:202–208, 1985.

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