Optimal Projective Algorithms for the List Update Problem

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

[1] Institute for Theoretical Computer Science, ETH Zürich, 8092 Zürich, Switzerland.
{ambuehl, gaertner}@inf.ethz.ch
[2] Mathematics Department, London School of Economics, London WC2A 2AE, United Kingdom. stengel@maths.lse.ac.uk

Abstract. The list update problem is a classical online problem, with an optimal competitive ratio that is still open, somewhere between 1.5 and 1.6. An algorithm with competitive ratio 1.6, the smallest known to date, is COMB, a randomized combination of BIT and TIMESTAMP. This and many other known algorithms, like MTF, are *projective* in the sense that they can be defined by only looking at any pair of list items at a time. Projectivity simplifies both the description of the algorithm and its analysis, and so far seems to be the only way to define a good online algorithm for lists of arbitrary length. In this paper we characterize all projective list update algorithms and show their competitive ratio is never smaller than 1.6. Therefore, COMB is a best possible projective algorithm, and any better algorithm, if it exists, would need a non-projective approach.

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 by accessing the requested item. 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* [12] where that cost is i. The goal is to keep access costs small by rearranging the items in the list. 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 $\mathit{OFF}(\sigma)$. If the online algorithm serves σ with cost $\mathit{ON}(\sigma)$, then it is called *c-competitive* if for a suitable constant b

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

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 [12, 13]. This is also the best possible competitiveness for any deterministic online algorithm for the list update problem [12]. Another deterministic algorithm that is also 2-competitive is TIMESTAMP due to Albers [1], which moves the requested item x in front of all items which have been requested at most once since the last request to x.

Randomized algorithms can perform better on average, as first shown by Irani [10, 11]. Such an algorithm is called c-competitive if

$$E\big[ON(\sigma)\big] \le c \cdot OFF(\sigma) + b,$$

where the expectation is taken over the randomized choices of the online algorithm. Randomization is useful only against the *oblivious adversary* [6] 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.

In this case, lower bounds for the competitive ratio are harder to find; the first nontrivial bounds are due to Karp and Raghavan, see the remark in [12]. A general technique is Yao's theorem [15]: 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 [9]. In the partial cost model, a lower bound of 1.5 is easy to find as only two items are needed. Teia [14] generalized this idea to prove the same bound in the full cost model, where long lists are needed. Ambühl, Gärtner and von Stengel [5] showed a lower bound of 1.50084 for lists with five items in the partial cost model, using game trees and a modification of Teia's approach. The optimal competitive ratio for the list update problem (in the partial cost model) is therefore between 1.50084 and 1.6, but the true value is as yet unknown.

The upper bound of 1.6 is the last link so far in a chain of results, starting with the observation that MTF acts too eagerly in moving items to the front. The better algorithms BIT, COUNTER, and RANDOM RESET move the requested item to the front or leave it at its position, depending on the number of previous requests to the currently requested item. The elegant BIT algorithm stores a data bit—initially set to a random value—with each item. The bit is flipped at each request, and the item is moved to the front of the list when the bit has been set to one. BIT is 1.75-competitive. The related RANDOM RESET algorithm has competitive ratio $\sqrt{3}$, about 1.73. This ratio is improved to the Golden Ratio $(1 + \sqrt{5})/2$, about 1.62, by treating each item with a randomized combination of TIMESTAMP and MTF [1].

The best randomized list update algorithm known to date is the 1.6-competitive algorithm COMB [2]. It serves the request sequence with probability 4/5 using BIT [12]. With probability 1/5, COMB treats the request sequence using TIMESTAMP.

With the exception of Irani's algorithm SPLIT [10, 11], all the specific list update algorithms mentioned above are *projective*, meaning that the relative order of any two items in the list only depends on previous requests to those items. (A simple example for a non-projective algorithm is TRANSPOSE, which moves the requested item just one position further to the front.) The main result of this paper is a proof that, surprisingly, 1.6 is the best possible competitive ratio attainable by a projective algorithm. As a tool, we develop an explicit characterization of deterministic projective algorithms in terms of two functions for every item, responsible for the “macro”- and the “micro”-behavior of the item.

This result is significant in several respects. First, it puts an end to the search for improved algorithms via combinations of existing projective ones. This approach has been used successfully in the past, as the results mentioned above indicate, but it has reached its limits with the development of the COMB algorithm. New and better algorithms (if they exist) have to be non-projective, and must derive from new, yet to be discovered, design principles. Second, the characterization of projective algorithms is a step forward in understanding the structural properties of list update algorithms. Under this characterization, the largest and so far most significant class of algorithms appears in a new, unified way. Third, our lower bound construction gives rise to an explicit test scenario for new algorithms: we construct a set of request sequences with the property that a randomly chosen instance from the set is “hard” for any projective algorithm. A new, supposedly better, algorithm should therefore be able to defeat those hard instances, and this might be more difficult than to defeat some ad-hoc set of instances.

2 Projective algorithms

Consider a list with n items. For a request sequence σ and two list items x and y, the *projection* of σ on the unordered pair $\{x, y\}$ is denoted by σ_{xy} and defined as the sequence obtained from σ by deleting all requests to items other than x or y. We write σ_x instead of σ_{xx}. For a given deterministic online algorithm, let $S(\sigma)$ denote the list state after the request sequence σ is served. List states are written as $[x_1 x_2 \dots x_n]$ where x_1 is the item at the front of the list. The list state projected to the pair $\{x, y\}$ is denoted by $S_{xy}(\sigma)$, which is either $[xy]$ or $[yx]$, indicating the relative position of x and y after σ is served.

A deterministic algorithm is projective if the relative order of any two items after any sequence σ does not depend on requests to the other items:

Definition 1 (Projective Algorithms). *A deterministic list update algorithm is projective if for all request sequences σ and any two list items x and y*

$$S_{xy}(\sigma) = S_{xy}(\sigma_{xy}). \tag{1}$$

A projective algorithm A can be analyzed in a much simpler way than a general one, since only two-item lists have to be studied [7]. If the projected cost on two items x, y of this algorithm is A_{xy} (with each request to x or y contributing either 0 or 1, depending on whether the requested item is before or behind the other item in the list), then its total cost is given by

$$A(\sigma) = \sum_{\{x,y\} \subseteq L} A_{xy}(\sigma_{xy}), \tag{2}$$

where L is the set of list items [2, 8]. Furthermore, the optimal offline cost OFF_{xy} of serving σ_{xy} is easy to describe, and gives a lower bound $\overline{OFF}(\sigma)$ defined similar to (2) for the optimal offline cost. This quantity is used to prove the competitive ratio of COMB and other algorithms. The simple equation (2) holds only in the partial cost model, which is therefore the natural choice when studying projective algorithms.

Projective algorithms have a natural generalization, where we demand the relative order of any k-tuple of list items to depend only on the requests to these k items. It turns out that for lists with more than k items, only projective algorithms satisfy this condition. This follows from the fact that e.g. for $k = 3$, $S_{xyz}(\sigma) = S_{xyz}(\sigma_{xyz})$ implies that the relative order of any pair from $\{x, y, z\}$ is independent of the requests to any item in $L \setminus \{x, y, z\}$. As soon as we have $k + 1$ list items, it is easy to see that the constraints for all k-tuples enforce the relative order of any pair of list items to be independent of the requests to other items.

We define a *randomized* online algorithm as projective if it is a (not necessarily finite) probability distribution over deterministic projective algorithms. A less restrictive definition is conceivable, but would not allow us to prove the lower bound for projective algorithms that we intend and that we think is useful. Namely, one could call a randomized online list update algorithm projective if serving any request sequence σ induces a distribution on list states $S_{xy}(\sigma)$ that only depends on σ_{xy}. To illustrate the problem with this definition, consider the following randomized algorithm on a list of two items only: If the current list state is $[xy]$ and y is requested following a request to x, move y to the front with probability 1/2, and if y is requested following a request to y (that is, y was not moved at the preceding request), then move y to the front with certainty. It is not hard to see that such a randomized online algorithm has competitive ratio 1.5, which is the best possible. Furthermore, any randomized algorithm showing this as projective behavior on a longer list would also be 1.5-competitive. It is indeed possible to construct such an algorithm for lists with up to four items using partial orders [3], but impossible for lists with five or more items, as recently proved by the authors [5].

In the following, we will characterize the deterministic projective algorithms in a way that makes their projective behavior transparent, and unifies many known algorithms. By our above assumption that considers a randomized projective algorithm as a probability distribution over deterministic ones, we will be able to use this characterization in the lower bound proof later.

3 Critical requests

Consider a given deterministic online list update algorithm that is projective according to Definition 1. In order to obtain a meaningful characterization of such an algorithm, we assume $n > 2$ since on lists with only two items, any algorithm is projective. Let $i, j > 0$ and consider request sequences σ with exactly i requests to x and j requests to y ($x \neq y$), that is, $\sigma_x = x^i$ (the i-fold repetition of x) and $\sigma_y = y^j$. Then we say x^i and y^j are *equivalent*, written $x^i \sim y^j$, if there are request sequences σ and σ' so that

$$\begin{aligned} \sigma_x = \sigma'_x = x^i, \;\; \sigma_y = \sigma'_y = y^j, \\ S_{xy}(\sigma) = [xy], \;\; S_{xy}(\sigma') = [yx]. \end{aligned} \tag{3}$$

In other words, one should be able to shuffle the requests to x and y such that either x or y is in front after serving the request sequence. Assume, for the moment, that (3) holds for any two items x and y and any $i, j > 0$ (we deal with the general case in the next section). Under this assumption, the algorithm can be characterized in terms of the central concept of *critical requests*.

For any list item x, let $F_x : \mathbb{N}^+ \to \mathbb{N}^+$ be a function so that $F_x(i) \leq i$ for all i. Then F_x defines the *critical request* to x in a request sequence σ as the $F_x(i)$th request to x in σ if $\sigma_x = x^i$. We say that the given algorithm *operates* according to these critical requests if, after serving any request sequence σ, the relative order of the items requested at least once is the reverse order of their critical requests in σ. In other words, x precedes y after σ with $\sigma_x = x^i$, $\sigma_y = y^j$, and $i, j > 0$ if and only if the $F_x(i)$th request of x was later than the $F_y(j)$th request to y. As a simple illustration, observe that MTF uses $F_x(i) = i$ for all $i > 0$ and $x \in L$. In the next section we will also deal with non-requested items. Paid exchanges can be represented by critical request functions that are not monotone.

There is also a natural "dual" way to deal with critical requests: At any time, an item precedes another if its critical request was *earlier*. In this case, we say that the algorithm operates *dually* according to critical requests. For example, operating dually on $F_x(i) = i$, for all i, results in the *move-to-back* algorithm. Although such behavior cannot be competitive, it defines a projective algorithm.

Any critical request functions F_x for the list items x therefore define two list update algorithms. The algorithms are projective since $F_x(i)$ does not depend on requests to items other than x. Furthermore, condition (3) holds for $i, j > 0$: sequences σ and σ' with $\sigma_{xy} = x^i y^j$ and $\sigma'_{xy} = y^j x^i$ will always result in $S_{xy}(\sigma) \neq S_{xy}(\sigma)$. The following theorem shows that, conversely, *any* projective list update algorithm fulfilling (3) arises from critical requests.

Theorem 1. *Let A be a projective algorithm on a list with n items, $n > 2$, so that for all items x, y and all $i, j > 0$, property (3) holds. Then A operates (or operates dually) according to suitable critical request functions.*

Proof. Assume that $i, j, k > 0$ and σ and σ' are request sequences (over only three items x, y and z) with $\sigma_x = \sigma'_x = x^i$, $\sigma_y = \sigma'_y = y^j$, and $\sigma_z = \sigma'_z = z^k$, so that

$$S_{xy}(\sigma_{xy}) = [xy] \quad \text{and} \quad S_{xz}(\sigma_{xz}) = [zx] \tag{4}$$

and

$$S_{xy}(\sigma'_{xy}) = [yx] \quad \text{and} \quad S_{xz}(\sigma'_{xz}) = [xz]. \tag{5}$$

Such sequences exist by assumption (3) and projectivity of A, and since the projections considered in these equations can be combined into sequences σ and σ' of requests to the three items.

Note that (4) and (5) imply $S_{yz}(\sigma) = [zy]$ and $S_{yz}(\sigma') = [yz]$, hence by projectivity

$$\sigma_{yz} \neq \sigma'_{yz}. \tag{6}$$

By labelling each request to an item with its position in the unary projection to that item (e.g. the fifth request to x will be labelled $x_{(5)}$), σ_{xy} and σ'_{xy} (and similarly σ_{xz} and σ'_{xz}) can be considered as permutations that may be transformed into each other by successively transposing pairs of consecutive requests.

We can even assume that σ_{xy} and σ'_{xy} differ only in a single such transposition, namely the (not necessarily unique) one responsible for the reversal of the list state S_{xy} during the transformation. Similarly, we assume that σ_{xz} and σ'_{xz} differ only in a single transposition of consecutive requests to x and z.

Suppose the transposition $\sigma_{xy} \to \sigma'_{xy}$ involves $x_{(q)}$ and $y_{(\ell)}$, while $x_{(r)}$ and $z_{(m)}$ participate in the transposition $\sigma_{xz} \to \sigma'_{xz}$. We now prove that $q = r$, which is also easily seen to imply that this value is well-defined: It neither depends on σ and σ' nor on the specific transposition we consider, but only on σ_x.

For this, assume $q \neq r$ and consider the sequence σ. W.l.o.g., $x_{(q)}$ and $y_{(\ell)}$ are consecutive requests in σ—otherwise we can transpose $y_{(\ell)}$ with all in-between requests to z without changing the projections to $\{x, y\}$ and $\{x, z\}$. Similarly, suppose that $x_{(r)}$ and $z_{(m)}$ are consecutive. A sequence σ' satisfying (5) is now obtained from σ by transposing both $x_{(q)}$ with $y_{(\ell)}$ and $x_{(r)}$ with $z_{(m)}$. Under this operation, however, the projection to $\{y, z\}$ remains invariant, a contradiction to (6). Hence, we must have $q = r$.

We have seen that for all items x and all i, there is a unique critical value $F_x(i) = q$, and it remains to show that A operates (or operates dually) according to the $F_x(i)$.

First of all, we need to show that the relative order of any two items only depends on the order of their critical requests. By the above arguments, whenever σ_{xy} and σ'_{xy} satisfy

$$S_{xy}(\sigma_{xy}) \neq S_{xy}(\sigma'_{xy}) \tag{7}$$

and differ only by a single transposition of consecutive requests, this transposition involves the critical request of x. By symmetry, it also involves the critical request of y. In general, when we transform σ_{xy} into σ'_{xy} by transposing consecutive requests, property (7) holds if and only if the two critical requests have been transposed an odd number of times, which fixes their relative order.

Now consider a request sequence σ over an n-item list such that $S(\sigma) = [x_1 x_2 \dots x_n]$. Let p_i be the position of x_i's critical request in σ. If we do not have $p_1 > p_2 > \cdots > p_n$ (A operates on F) or $p_1 < p_2 < \cdots < p_n$ (A operates dually on F), we must have an index i such that either $p_i < p_{i+1} > p_{i+2}$ or $p_i > p_{i+1} < p_{i+2}$. In both cases, we can manipulate σ such that the critical requests of x_i and x_{i+2} change their order, but both keep their relative order w.r.t. the critical request of x_{i+1}. In the list obtained after serving σ, items x_i and x_{i+2} change their relative order under this manipulation, while they keep their relative order w.r.t. x_{i+1}. This is impossible. $\square$

The assumption of at least three list items in the preceding theorem is crucial. On lists with only two items, any algorithm is projective, but cannot always be defined in terms of critical requests. This follows from cardinality considerations: There are $\binom{i+j}{i}$ request sequences with i requests to x and j requests to y, each of which can have its own list state after being served, but only $i \cdot j$ many ways of defining critical requests. As an example, the algorithm that puts the second-to-last requested item at the front of the two-item list cannot be defined in terms of critical requests.

4 Containers

Not all projective algorithms fulfill condition (3) for all x^i and y^j. As an example, consider the algorithm where all items requested an odd number of times precede all items requested an even number of times, and where the items within each of these two sets are arranged according to the MTF rule. Then (3) fails if i is odd and j is even or vice versa. According to the general characterization of projective algorithms that we will give in this section, the odd- and even-requested items in this example form separate "containers" that represent sublists of the list. Within one container, items are moved according to critical requests, but items in different containers are always in one and the same relative position.

To make this precise, consider a given projective list update algorithm and the set

$$U = \{x^i \mid i \in \mathbb{N},\ x \in L\}$$

of unary projections of request sequences, where L denotes the set of items in the list.

Recall that we write $x^i \sim y^j$ whenever (3) holds. It makes sense to allow $i = 0$ and $j = 0$ as well. By generalizing (3) we get $x^0 \not\sim y^j$ for all $x \neq y$ and $j \in \mathbb{N}$.

For x, y, z distinct, if $x^i \sim y^j$ and $y^j \sim z^k$, then, by projectivity, there are always two sequences σ and σ' containing the three unary projections such that $S(\sigma) = [xyz]$ and $S(\sigma') = [zyx]$, which implies $x^i \sim z^k$. It is easy to see that $\sim$ is an *equivalence relation* on U if we stipulate $x^i \sim x^i$ for any $x^i \in U$ and also $x^i \sim x^j$ for $i \neq j$ if and only if there is a z^k with $z \neq x$ such that $x^i \sim z^k$ and $z^k \sim x^j$.

An equivalence class under $\sim$ shall be called a *container*. Let $\mathcal{C}$ denote the set of containers and $c_x(i)$ denote the container containing x^i. If x^i and y^j are in different containers, we write $c_x(i) < c_y(j)$ whenever for some σ with $\sigma_x = x^i$ and $\sigma_y = y^j$ we have $S_{xy}(\sigma) = [xy]$ (and hence for all such σ, since (3) does not hold). It is easy to see that this does not depend on the choice of the representatives x^i and y^j from each container. A special case occurs if both x^i and x^j are the only members in their respective containers. In this case, no canonical order exists, and we set $c_x(i) < c_x(j)$ if $i < j$.

Then $<$ defines a *total order* on the containers, which has a natural interpretation: After a request sequence σ, consider the unary projections σ_x for each item x, and their corresponding containers. If two projections x^i and y^j are in different containers, x is in front of y if and only if $c_x(i) < c_y(j)$. Hence the containers represent sublists of the list state $S(\sigma)$, and we will say that $c_x(i)$ contains the item x if $\sigma_x = x^i$.

This characterizes any projective algorithm, apart from its behavior within each container, which is easy to describe: If there is only one item in the container, the position of the item is that of the container. A container containing only unary projections for two distinct items can have an arbitrary behavior of its items, since any algorithm on only two items is projective. If the container contains projections for at least three items, Theorem 2 applies, that is, the algorithm operates or operates dually according to suitable critical requests defined for the unary projections in that container. For this, observe that by definition of $\sim$, all empty projections x^0 are in a container of their own, so whenever a container has at least two items, each item has been requested at least once, in which case the critical requests exist.

To summarize, we can express any deterministic projective algorithm by a pair of functions $c_x : \mathbb{N} \to \mathcal{C}$ and $F_x : \mathbb{N}^+ \to \mathbb{N}^+$ for all $x \in L = \{x_1, x_2, \ldots, x_n\}$. The ordering of the values $c_x(0)$ corresponds to the initial list state. In the following examples, we denote the containers by integers, where the ordering of the integers corresponds to the ordering of the containers.

MTF moves all items into a common container 0 at their first request. All items use $c_{x_i} \equiv (i, 0, 0, 0, \ldots)$ and $F_{x_i}(k) = k$.

TIMESTAMP moves all items into a common container 0 at their second request. By definition of $\sim$, an item cannot stay in its initial container after the first request, so all items will use $c_{x_i} \equiv (2i, 2i-1, 0, 0, 0, \ldots)$ and $F_{x_i}(k) = \max(1, k-1)$.

FREQUENCY COUNT makes heavy use of containers. Items are ordered according to the number of requests to them, so all items requested k times are in container $-k$. The functions used are $c_{x_i} \equiv (i, -1, -2, \ldots)$ and $F_{x_i}(k) = k$.

BIT is a randomized algorithm. In the beginning, every item tosses a fair coin to decide whether it uses the pair $(F^0_{x_i}, c^0_{x_i})$ or $(F^1_{x_i}, c^1_{x_i})$ with $c^0_{x_i} \equiv (2i, 2i-1, 0, 0, 0, \ldots)$, $F^0_{x_i} \equiv (1, 1, 3, 3, 5, 5, \ldots)$, $c^1_{x_i} \equiv (2i, 0, 0, 0, \ldots)$, $F^1_{x_i} \equiv (1, 2, 2, 4, 4, 6, 6, \ldots)$.

5 Lower bound

In this section, we use the characterization of projective algorithms from the previous section to prove that no such algorithm is better than 1.6-competitive. Algorithms with a good competitive ratio never operate dually according to critical requests, and have the critical request close to the last request, for every item. That is, they fulfill $i - F_x(i) \leq 1$ most of the time. Therefore we work with $f_x(i) = i - F_x(i)$ in the following. Recall that MTF is defined by $f_x(i) = 0$, and TIMESTAMP by $f_x(i) = 1$.

Motivated by this discussion, we consider projective algorithms A for lists of more than two items that fulfill the following additional assumptions:

(i) A constant p exists such that after at most p requests to every item, all items will reside in a single common container, and A operates (i.e. does not operate dually) according to critical requests within that container, and

(ii) the values $f_x(i)$ that determine the critical requests can be uniformly bounded by some constant M, where w.l.o.g. $M \geq 3$.

Let us call an algorithm satisfying (i) and (ii) *regular*.

Given any $\varepsilon > 0$ and b, we will show that there is a probability distribution π on a finite set Λ of request sequences so that

$$\sum_{\lambda \in \Lambda} \pi(\lambda) \frac{A(\lambda)}{\mathit{OFF}(\lambda) + b} \geq 1.6 - \varepsilon, \tag{8}$$

for any deterministic regular algorithm A. Then *Yao's theorem* [15] asserts that also any randomized regular algorithm has competitive ratio $1.6-\varepsilon$ or larger. This holds for any fixed p and M. Hence the competitive ratio is at least 1.6. This is achieved by COMB and therefore a tight bound for projective algorithms.

The same holds for general projective algorithms, but we defer the technicalities of that to the full paper (see also section 6).

All $\lambda \in \Lambda$ will consist of only two items x and y. With the constant M from (ii), let

$$\phi = x^M \; yxyx^M \; yx^M \; y^M \; xyxy^M \; xy^M \; x^M \; y^M. \tag{9}$$

ϕ consists of eight *blocks*, each of which ends in x^M or y^M. Let $H = |\phi|/2$, and let k and T be arbitrary positive integers. Then the set of sequences in (8) is given by

$$\Lambda = \{x^t x^{3+h} y^{3+h} \phi^k \mid 0 \le h < H,\ 0 \le t < T\}, \tag{10}$$

where any λ in Λ is chosen with equal probability $1/HT$ by π.

First, observe that *OFF* pays ten units for each repetition of ϕ (which always starts in offline list state $[yx]$), and therefore all sequences in Λ have equal offline cost $1 + 10k$. This and the fact that $\pi(\lambda)$ for $\lambda \in \Lambda$ is constant allows us to rewrite (8) as

$$\frac{\sum_{\lambda\in\Lambda} A(\lambda)}{\sum_{\lambda\in\Lambda} (\mathit{OFF}(\lambda) + b)} > 1.6 - \varepsilon. \tag{11}$$

The offline cost $\mathit{OFF}(\lambda)$ in (11), as well as the online cost $A(\lambda)$, can grow arbitrarily large with k, so that we can assume w.l.o.g. that $b = 0$ in (11), adapting ε suitably.

In the rest of this section we show that (10) yields the desired property (11). We say that A is in *state* (i, j) if it has served σ with $\sigma_x = x^i$ and $\sigma_y = y^j$, where σ is some prefix of a sequence λ in Λ. That sequence σ is a random variable since the particular order of requests to x and y is usually not known.

We say that a sequence λ in Λ *switches from* x in state (i, j) if λ has the prefix σ with $\sigma_x = x^i$ and $\sigma_y = y^j$ and σ ends in x^M. Similarly, we say that λ switches from y in state (i, j) if that prefix σ ends in y^M. A state (i, j) is called *good* if it fulfills the following conditions:

(a) there are four sequences in Λ that switch from x in state (i, j). They continue with the requests y^M, y^M, yx^M, and $yxyx^M$, respectively;

(b) the same holds with x and y interchanged;

(c) properties (a) and (b) also hold for the states $(i-1, j)$ and $(i, j-1)$.

This means, for every good state (i, j) and each of the eight blocks in ϕ, there is exactly one sequence σ in Λ that starts with this block in state (i, j). Let $A(i, j)$ be the sum of costs incurred by A on those eight blocks, and $\mathit{OFF}(i, j)$ the corresponding sum of offline costs. In general, $A(i, j)$ denotes the sum of costs incurred by A on all next blocks of the (at most eight) sequences switching from x or y in state (i, j), and $\mathit{OFF}(i, j)$ is the corresponding offline cost.

We will show that for every good state (i,j), $A(i,j)$ is at least 16, while $OFF(i,j)$ is always 10. Together with the facts that most states are good states (which we will prove below), and that the cost for serving the initial prefix $x^t x^{3+h} y^{3+h}$ is independent of k and can therefore be neglected for large k, this gives

$$\frac{\sum_{\lambda \in \Lambda} A(\lambda)}{\sum_{\lambda \in \Lambda} OFF(\lambda)} = \frac{\sum\limits_{(i,j)} A(i,j) + \sum\limits_{h,t} A(x^t x^{3+h} y^{3+h})}{\sum\limits_{(i,j)} OFF(i,j) + \sum\limits_{h,t} OFF(x^t x^{3+h} y^{3+h})}$$

$$\approx \frac{\sum\limits_{(i,j)\ \text{good}} A(i,j)}{\sum\limits_{(i,j)\ \text{good}} OFF(i,j)} \geq \min_{(i,j)\ \text{good}} \frac{A(i,j)}{OFF(i,j)},$$

thus proving the lower bound, because the minimum is at least $16/10 = 1.6$.

By considering each of the four continuing sequences y^M, y^M, yx^M, and $yxyx^M$ in (a), we see that the sum of their offline costs is five. Hence, we have to show that the sum of online costs is at least eight. This is not always the case: It is possible that a certain choice of the critical request functions will result in an online cost of only seven units. However, we will show that those particular critical requests will incur nine units of online cost in state $(i-1,j)$, one of which we can "borrow" for $A(i,j)$. This results in eight units of online cost, for all good states.

Consider a sequence in a good state (i,j) that, as in (a), switches from x, so that the next requests are y^M, y^M, yx^M, and $yxyx^M$ (our reasoning will then apply to (b) by symmetry). Since the first two of these requests are yy, yy, yx, yx, their total online cost is six units: By assumption (ii), the critical request to x is among the preceding requests x^M. Hence four units are to be paid for the first request to y, and then two extra units, namely on the request to x in yx and yx if $f_y(j+1) = 0$, and on the second request to y in yy and yy if $f_y(j+1) \geq 1$. A seventh unit is spent in serving the second request to y or to x in $yxyx^M$. The only case where no more than these seven online cost units occur is if

$$f_x(i+1) = 0 \qquad \text{and} \qquad f_y(j+2) = 1. \tag{12}$$

Namely, $f_y(j+2) \leq 1$ is necessary since otherwise y would not be in front of x at the third request to y in the sequences y^M, adding at least two more cost units. If $f_y(j+1) = 0$, then we need $f_x(i+1) = 0$ to avoid another cost unit when serving the second request to x in yx^M, and hence $f_y(j+2) \geq 1$ to avoid that y is moved to the front at the second request to y in $yxyx^M$ since that would create an extra cost unit for serving the second x. If $f_y(j+1) \geq 1$, then we need again $f_x(i+1) = 0$ and $f_y(j+2) \geq 1$ to avoid that y is moved to the front at the second request to y in $yxyx^M$. Together, this implies (12). Hence, whenever (12) fails, the online algorithm incurs eight or more unit costs.

The seven online cost units in case (12) create *nine* online cost units in state $(i-1,j)$ for the sequences switching from y. Namely, by (c), the subsequent requests are x^M,

x^M, xy^M, and $xyxy^M$. As before, six online units will be spent on the first two of these requests which are xx, xx, xy, xy, and a seventh unit either on the second y in xy^M or on the second x in $xyxy^M$. That last sequence, however, will incur two additional cost units: Because of (12), x is in front of y after the third and forth request in $xyxy^M$, causing two more units for serving the second and third requests to y.

It remains to show that most states are good. The sequences λ in Λ are, by (10), essentially k-fold repetitions of ϕ.

The random initial subsequence $x^{3+h}y^{3+h}$ of λ means that all pairs of states (i,j) and $(i+1,j+1)$, apart from those with low or high values of i and j, are equally likely reached by any request in ϕ. Note that $|\phi_x| = |\phi_y|$. The additional prefix x^t of λ does the same for the pairs of states (i,j) and $(i+1,j)$ except for those with small or large values of i and j. Figure 1 displays the good and the bad states in a diagram. There are $\Theta(kHT)$ good states, but only $\Theta(kH^2)$ bad ones. By choosing T large enough, the contribution of bad states can be neglected, so that (8) holds.

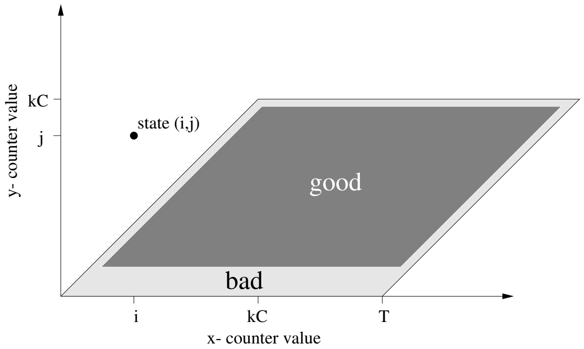

Fig. 1. xy-diagram

We have proved a lower bound of 1.6 for the competitive ratio of any *regular* projective algorithm A, defined by a probability distribution over deterministic regular algorithms. By definition, this means that A is using only one active container in the long run, and that the distance between current and critical request is bounded for all items. Because of lack of space, we will deal with the other cases only in the full version. We thus obtain

Theorem 2. *Any projective list update algorithm has a competitive ratio of at least* 1.6 *in the partial cost model, and this bound is best possible, as the algorithm COMB demonstrates.*

6 Conclusion

An open problem is to extend this result to the full cost model, even though this model is not very natural in connection with projective algorithms. This would require request sequences over arbitrarily many items, and it is not clear whether an approach similar to the one given here can work.

Another ambitious goal is to further improve the lower bound in case of non-projective algorithms. Here, the techniques of the paper do not apply at all, and to get improvements that are substantially larger than the ones obtainable with the methods of [5] requires substantial new insights.

Finally, the search for good non-projective algorithms has become an issue with our result. Irani's SPLIT algorithm [10, 11] is the only one known of this kind with a competitive ratio below 2. A major obstacle for finding such algorithms is the difficulty of their analysis, because pairwise methods are not applicable, and other methods (e.g. the potential function method) have not been studied in depth. We hope that our result can stimulate further research in this direction.

References

1. S. Albers (1998), Improved randomized on-line algorithms for the list update problem. *SIAM J. Comput.* 27, no. 3, 682–693 (electronic). Preliminary version in *Proc. 6th Annual ACM-SIAM Symp. on Discrete Algorithms* (1995), 412–419.
2. S. Albers, B. von Stengel, and R. Werchner (1995), A combined BIT and TIMESTAMP algorithm for the list update problem. *Inform. Process. Lett.* 56, 135–139.
3. S. Albers, B. von Stengel, and R. Werchner (1996), List update posets. Manuscript.
4. S. Albers and J. Westbrook (1998), Self Organizing Data Structures. In A. Fiat, G. J. Woeginger, "Online Algorithms: The State of the Art", *Lecture Notes in Comput. Sci.*, 1442, Springer, Berlin, 13–51.
5. C. Ambühl, B. Gärtner, and B. von Stengel (2000), A new lower bound for the list update problem in the partial cost model. To appear in *Theoret. Comput. Sci.*
6. 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.
7. J. L. Bentley, and C. C. McGeoch (1985), Amortized analyses of self-organizing sequential search heuristics. *Comm. ACM* 28, 404–411.
8. A. Borodin and R. El-Yaniv (1998), Online Computation and Competitive Analysis. Cambridge Univ. Press, Cambridge.
9. 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.
10. S. Irani (1991), Two results on the list update problem. *Inform. Process. Lett.* 38, 301–306.
11. S. Irani (1996), Corrected version of the SPLIT algorithm. Manuscript.
12. N. Reingold, J. Westbrook, and D. D. Sleator (1994), Randomized competitive algorithms for the list update problem. *Algorithmica* 11, 15–32.
13. D. D. Sleator, and R. E. Tarjan (1985), Amortized efficiency of list update and paging rules. *Comm. ACM* 28, 202–208.
14. B. Teia (1993), A lower bound for randomized list update algorithms, *Inform. Process. Lett.* 47, 5–9.
15. A. C. Yao (1977), Probabilistic computations: Towards a unified measure of complexity. *Proc. 18th FOCS*, 222–227.