Optimal Lower Bounds for Projective List Update Algorithms*

Christoph Ambühl† Bernd Gärtner‡ Bernhard von Stengel§

March 8, 2012

Abstract

The list update problem is a classical online problem, with an optimal competitive ratio that is still open, known to be 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 the TIMESTAMP algorithm TS. This and almost all other list update algorithms, like MTF, are *projective* in the sense that they can be defined by looking only at any pair of list items at a time. Projectivity (also known as "list factoring") 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 that their competitive ratio is never smaller than 1.6 in the partial cost model. Therefore, COMB is a best possible projective algorithm in this model.

Keywords: linear lists, online algorithms, competitive analysis.

AMS subject classifications: 68W27, 68W40, 68P05, 68P10.

*We thank two referees for helpful comments. A preliminary version of this paper appeared in [6].

†Dalle Molle Institute for Artificial Intelligence (IDSIA), 6928 Manno, Switzerland. Supported by the Swiss National Science Foundation project 200020-122110/1 "Approximation Algorithms for Machine Scheduling Through Theory and Experiments III" and by Hasler Foundation Grant 11099. Email: christoph.ambuhl@googlemail.com

‡Institute of Theoretical Computer Science, ETH Zürich, 8092 Zürich, Switzerland. Email: gaertner@inf.ethz.ch

§Department of Mathematics, London School of Economics, London WC2A 2AE, United Kingdom. Email: stengel@nash.lse.ac.uk

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* [14] 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 $\mathtt{OPT}(\sigma)$. If the online algorithm serves σ with cost $\mathtt{A}(\sigma)$, then it is called *c-competitive* if for a suitable constant b

$$\mathtt{A}(\sigma) \leq c \cdot \mathtt{OPT}(\sigma) + b \qquad (1)$$

for all request sequences σ and all initial list states. The infimum over all c so that (1) holds for $\mathtt{A}$ is also called the *competitive ratio* of $\mathtt{A}$. If the above inequality holds even for $b = 0$, the algorithm $\mathtt{A}$ is called *strictly c-competitive* [9].

The *move-to-front* rule MTF, for example, which moves each item to the front of the list after it has been requested, is strictly 2-competitive [14, 15]. This is also the best possible competitiveness for any deterministic online algorithm for the list update problem [14]. Another 2-competitive deterministic algorithm is TS, which is the simplest member of the TIMESTAMP class due to Albers [1]. TS moves the requested item x in front of all items which have been requested at most once since the last request to x.

As shown first by Irani [13], *randomized* algorithms can perform better on average. Such an algorithm is called c-competitive if

$$E[\mathtt{A}(\sigma)] \leq c \cdot \mathtt{OPT}(\sigma) + b,$$

for all σ and all initial list states, where the expectation is taken over the randomized choices of the online algorithm; this definition implies that the request sequences σ are generated by an *oblivious adversary* that does not observe the choices of the online algorithm. 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 the algorithm BIT [14]. With probability $1/5$, COMB treats the request sequence using TS.

Lower bounds for the competitive ratio of randomized algorithms are harder to find; the first nontrivial bounds are due to Karp and Raghavan, see the remark in [14]. In the partial cost model, a lower bound of 1.5 is easy to find as only two items are needed. Teia [16] generalized this idea to prove the same bound in the full cost model, which requires long lists. The authors [7] showed a lower bound of 1.50084 (improved to 1.50115 in [5, p. 38]) 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 1.50115 and 1.6, but the true value is as yet unknown.

With the exception of Irani's algorithm `SPLIT` [13], all the specific list update algorithms mentioned above are *projective*, meaning that the relative order of any two items x and y in the list after a request sequence σ only depends on the initial list state and the requests to x and y in σ. The simplest example of a projective algorithm is `MTF`. In order to determine whether x is in front of y after σ, all that matters is whether the last request to x was before the last request to y. The requests to other items are irrelevant.

A simple example of a non-projective algorithm is `TRANSPOSE`, which moves the requested item just one position further to the front.

Projection to pairs of items, also known as "list factoring", is the main tool for analyzing list update algorithms. It has also been applied recently to other performance models of list processing [10, 11].

Our results. The main result of this paper (Theorem 18) states that 1.6 is the best possible competitive ratio attainable by a projective algorithm. As a tool for proving it, we develop an explicit characterization of deterministic projective algorithms.

These results are significant in two respects. First, they show that the successful approach of combining existing projective algorithms to obtain improved ones has reached its limit 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. With this characterization, the largest and so far most significant class of algorithms appears in a new, unified way.

The complete characterization of projective algorithms turns out to be rather involved. However, there is a simple subclass of projective algorithms which already covers all projective algorithms that can be expected to have a good competitive ratio. We call them *critical request algorithms*. A list update algorithm is completely described by the list state after a request sequence σ has been served; this can be done because we can assume that all changes in the list state are due to paid exchanges, as explained in further detail at the beginning of the next section. For critical request algorithms, the *unary projections* to individual items suffice to describe that list state. For a request sequence σ and list item x, deleting all requests to other items defines the unary projection σ_x, which is an i-fold repetition of requests to x, written as x^i, for some $i \geq 0$. In Section 5 it will be necessary to consider unary projections x^0 and y^0 of length zero as different if the items x and y are different; for the moment, this distinction does not matter. With L as the set of list items, let the set of these unary projections be

$$U = \{x^i \mid x \in L,\ i \geq 0\}. \tag{2}$$

Definition 1 (Critical request algorithm).
A *deterministic* critical request algorithm is defined by a function

$$F : U \to \{0, 1, 2, \ldots\}, \qquad \text{with } F(x^i) \leq i \text{ for any } x \in L,\ i \geq 0.$$

We call the $F(\sigma_x)$th request to x in σ the *critical request* to x. If $F(\sigma_x)$ is zero (for example if σ_x is the *empty sequence* ϵ), then x has no critical request. In the list state after σ, all items with a critical request are grouped together in front of the items without critical request. The items with critical requests are ordered according to the time of the $F(\sigma_x)$th request to x in σ. The later a critical request took place in the sequence, the closer the item is to the front. The items without critical request are placed at the end of the list according to their order in the initial sequence. A *randomized critical request algorithm* is a discrete probability distribution on the set of deterministic critical request algorithms. □

As an example, consider the online algorithm for three items a, b, and c with the function F shown in the following table for requests up to four items.

i	0	1	2	3	4
$F(a^i)$	0	1	0	2	2
$F(b^i)$	0	0	2	2	4
$F(c^i)$	0	1	2	2	2

In the rest of this paper, list states are written as $[x_1 x_2 \dots x_n]$ where x_1 is the item at the front of the list. Let the initial list state be $[abc]$. Consider the list state after $\sigma = abbcab$. We have $F(\sigma_a) = F(aa) = 0$, hence a does not have a critical request. For b we have $F(\sigma_b) = F(bbb) = 2$, therefore the second request to b in σ is its critical request. For c we have $F(\sigma_c) = F(c) = 1$. Thus after σ, the list state is $[cba]$. If we augment σ by another request to a, item a moves to the front, because its critical request is the second.

Algorithms based on critical request functions are clearly projective, since the relative order of any pair of items just depends on the relative order of the requests to x and y in σ and the relative order of x and y in the initial list state.

In good online algorithms, the critical requests are very recent, like in MTF which is described by the critical request function $F(x^i) = i$ for all items x. We define the critical request *relative* to the current position by

$$f(x^i) = i - F(x^i), \tag{3}$$

from which the critical request function is recovered as $F(x^i) = i - f(x^i)$. Then MTF is given by $f(x^i) = 0$. Algorithm TS is described by $f(x^i) = 1$ for all items and all $i > 0$ (and $f(\epsilon) = 0$). Because the BIT algorithm [14] is randomized, its critical requests are also randomized. For every item x, its relative critical request function can be written as $f(\epsilon) = 0$ and, for $i > 0$,

$$f(x^i) = (b_x + i) \mod 2 \tag{4}$$

where $b_x \in \{0, 1\}$ is chosen once uniformly at random; so for a list with n items, BIT is the uniform distribution over 2^n different deterministic algorithms. For BIT, the critical request is the last or the second-to-last request with equal probability.

The structure of the paper is as follows. In the next section, we explain projective algorithms in more detail and how they can be analyzed. In Section 3, we give a characterization of so-called M-regular projective algorithms, followed by the lower bound of 1.6 for this important class of algorithms in Section 4. In Section 5, we characterize projective algorithms completely. We extend the lower bound to the full class in Section 6.

2 Projective Algorithms

In order to characterize list update algorithms, we first simplify their formal definition. The standard definition (of the partial cost model) considers a list state and a sequence of requests. For each request to one of the items of the list, the item can be accessed with access cost $i-1$ if the item is in position i, and then moved free of charge closer to the front. In addition, paid exchanges are allowed which can be applied both before and after accessing the item, at a cost of one unit for exchanging any two consecutive items.

Contrary to the claim of [15, Theorem 3], paid exchanges may strictly improve costs. For example, let the initial list state be $[abc]$ and $\sigma = cbbc$. Then an optimal algorithm moves a behind b and c before the first request to c. This requires paid exchanges.

In order to simplify the description of a list update algorithm, we assume that it operates using only paid exchanges, as follows: The list is in a certain state. The algorithm is informed about the next request, and then performs a number of paid exchanges, and is charged for their cost. It then accesses the requested item at cost $i-1$ when the item is the ith item in the list, without any further changes to the list. This mimicks free exchanges as well: Instead of first paying k units in order to access item x and then moving it at no charge t positions closer to the front, one can first move the item t positions forward and then access the item at cost $k-t$. In both cases, one pays exactly k units.

The above description ignores paid exchanges immediately before learning the next request; if the algorithm performs them after the request, it has only more information. In addition, paid exchanges also allow the transposition of items at unrealistic low costs that are behind the requested item in the list. This does not matter for our lower bound considerations.

The above considerations lead to a simplified but still equivalent model of list update algorithms: Any deterministic online algorithm $\mathtt{A}$ is specified by a function

$$S^{\mathtt{A}} : \Sigma \to \mathcal{L}.$$

Here, Σ denotes the set of finite request sequences (including the empty sequence ϵ), and $\mathcal{L}$ denotes the set of the $n!$ states the list of n items can attain. By definition, $S^{\mathtt{A}}(\sigma)$ is the list state after the last request of σ has been served by algorithm $\mathtt{A}$.

Consider a request sequence σ and assume it is followed by a request to item x, the extended sequence denoted by σx. Then the cost of serving request x is defined by: the cost of re-arranging the list from state $S^{\mathtt{A}}(\sigma)$ to $S^{\mathtt{A}}(\sigma x)$ by paid exchanges, plus the cost of accessing x in state $S^{\mathtt{A}}(\sigma x)$.

Using this notation, the initial list state can be denoted by $S^{\mathtt{A}}(\epsilon)$. We will omit the superscript A in $S^{\mathtt{A}}(\sigma)$ when the algorithm used is determined by the context.

In order to describe projective algorithms, we have to introduce the concept of *projections* of request sequences and list states. Let a request sequence σ be given and fix a pair of items x, y. The projection σ_{xy} of σ to x and y is the request sequence σ where all requests which are not to x or y are removed. Similarly, σ_x is σ with all requests other than those to x removed.

Given a list state L, the projection to x and y is obtained by removing all items except for x and y from the list. This is denoted by L_{xy}.

Definition 2. Let $S_{xy}(\sigma)$ be the projection of $S(\sigma)$ to x and y. A deterministic algorithm A is called *projective* if for all pairs of items x, y and all request sequences σ

$$S_{xy}(\sigma) = S_{xy}(\sigma_{xy}). \tag{5}$$

A randomized algorithm is projective if all deterministic algorithms that it chooses with positive probability are projective.

For any list update algorithm A, define the *projected cost* $\mathtt{A}_{xy}(\sigma)$ that A serves a request sequence σ, projected to the pair x, y, as follows: Consider all requests z in σ with corresponding prefix $\sigma' z$ of σ. Then $\mathtt{A}_{xy}(\sigma)$ is the number of times where $S_{xy}(\sigma')$ and $S_{xy}(\sigma' z)$ differ (which counts the necessary paid exchanges of x and y; this may happen even if $z \notin \{x, y\}$ in case A is not projective), plus the number of times where $z = x$ and $S_{xy}(\sigma' z) = [yx]$ or $z = y$ and $S_{xy}(\sigma' z) = [xy]$. □

Thus, an algorithm is projective if the relative position of any pair of items depends only on the initial list state and the requests to x and y in the request sequence.

Projective algorithms have a natural generalization, where the relative order of any k-tuple of list items depends 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, for example for $k = 3$, $S_{xyz}(\sigma) = S_{xyz}(\sigma_{xyz})$ (so the relative position of x and y does not depend on requests to w), and $S_{xyw}(\sigma) = S_{xyw}(\sigma_{xyw})$ (so the relative position of x and y does not depend on requests to z), which implies that $S_{xy}(\sigma)$ depends only on σ_{xy}.

Already in [8], Bentley and McGeoch observed that MTF is projective: Item x is in front of y if and only if y has not been requested yet or if the last request to x took place after the last request to y.

With the exception of Irani's SPLIT algorithm [13], projective algorithms are the only family of algorithms that have been analyzed so far, typically using the following theorem, for example in [1, 2, 9].

Theorem 3. If a (strictly) projective algorithm is c-competitive on lists with two items, then it is also (strictly) c-competitive on lists of arbitrary length.

Proof. Consider first an arbitrary list update algorithm $\mathtt{A}$. Let L be the set of list items. Then

$$\mathtt{A}(\sigma) = \sum_{\{x,y\}\subseteq L} \mathtt{A}_{xy}(\sigma), \tag{6}$$

because the costs $\mathtt{A}(\sigma)$ are given by the update costs for changing $S(\sigma')$ to $S(\sigma' z)$, which is the sum of the costs of paid exchanges of pairs of items, plus the cost of accessing z in state $S(\sigma' z)$.

For a projective algorithm $\mathtt{A}$ the relative behavior of a pair of items is, according to (5), independent of the requests to other items. It is therefore easy to see that $\mathtt{A}_{xy}(\sigma) = \mathtt{A}_{xy}(\sigma_{xy})$ for projective algorithms: Because $\mathtt{A}$ is projective, $\mathtt{A}_{xy}(\sigma_{xy})$ is also the cost of $\mathtt{A}$ for serving σ_{xy} on the two-item list containing x and y starting from $S_{xy}(\epsilon)$.

For the algorithm $\mathtt{OPT}$, the term $\mathtt{OPT}_{xy}(\sigma_{xy})$ is the cost of optimally serving σ_{xy} on the two-item list $S_{xy}(\epsilon)$. Hence, $\mathtt{OPT}_{xy}(\sigma) \geq \mathtt{OPT}_{xy}(\sigma_{xy})$. Then

$$\mathtt{OPT}(\sigma) = \sum_{\{x,y\}\subseteq L} \mathtt{OPT}_{xy}(\sigma) \geq \sum_{\{x,y\}\subseteq L} \mathtt{OPT}_{xy}(\sigma_{xy}) =: \overline{\mathtt{OPT}}(\sigma). \tag{7}$$

Let $\mathtt{A}$ be a projective algorithm that is c-competitive on two items. Then for every pair of items x, y there is a constant b_{xy} such that for all σ

$$\mathtt{A}_{xy}(\sigma_{xy}) \leq c \cdot \mathtt{OPT}_{xy}(\sigma_{xy}) + b_{xy}.$$

Then

$$\begin{aligned}
\mathtt{A}(\sigma) &= \sum_{\{x,y\}\subseteq L} \mathtt{A}_{xy}(\sigma_{xy}) \\
&\leq \sum_{\{x,y\}\subseteq L} (c \cdot \mathtt{OPT}_{xy}(\sigma_{xy}) + b_{xy}) \\
&\leq c \cdot \overline{\mathtt{OPT}}(\sigma) + \sum_{\{x,y\}\subseteq L} b_{xy} \\
&= c \cdot \overline{\mathtt{OPT}}(\sigma) + b \\
&\leq c \cdot \mathtt{OPT}(\sigma) + b.
\end{aligned}$$

For the strict case, just set all $b_{xy} := 0$. □

Not all algorithms are projective. Let $\mathtt{LMTF}$ be the algorithm that moves the requested item x in front of all items which have not been requested since the previous request to x, if there has been such a request.

It is easy to prove that on lists with two items, combining $\mathtt{LMTF}$ and $\mathtt{MTF}$ with equal probability would lead to a 1.5-competitive randomized algorithm. Obviously, if $\mathtt{LMTF}$ was projective, this bound would hold for lists of arbitrary length.

However, $\mathtt{LMTF}$ is not projective. This can be seen from the request sequence $\sigma = baacbc$ with initial list $L_0 = [abc]$. It holds that $S^{\mathtt{LMTF}}(\sigma) = cab$, whereas $S^{\mathtt{LMTF}}(\sigma_{bc}) = S^{\mathtt{LMTF}}(bcbc) = bca$. Hence $S^{\mathtt{LMTF}}_{bc}(\sigma) \neq S^{\mathtt{LMTF}}_{bc}(\sigma_{bc})$.

3 Critical Requests and M-regular Algorithms

In this section, we provide technical preliminaries for our results, and introduce M-regular algorithms, which move an item to the front of the list when it has been requested M times in succession.

Throughout this section, we consider deterministic projective list update algorithms. In order to refer to the individual requests to an item x, we write unary projections as

$$x^i = x_{(1)}x_{(2)} \dots x_{(i)},$$

that is, $x_{(q)}$ is the qth request to x in σ if $\sigma_x = x^i$, for $1 \leq q \leq i$.

Let $\mathcal{P}(\sigma)$ be the set of all permutations of the sequence σ. In particular, $\mathcal{P}(x^i y^j)$ consists of all sequences with i requests to x and j requests to y.

Swapping two requests $x_{(q)}$ and $y_{(l)}$ in a request sequence σ means that $x_{(q)}$ and $y_{(l)}$, which are assumed to be adjacent, change their position in σ. If two requests are not adjacent, they cannot be swapped.

Definition 4. Consider a deterministic projective list update algorithm $\mathtt{A}$. A pair of unary projections x^i, y^j is called *agile* if there exist two request sequences τ and τ' in $\mathcal{P}(x^i y^j)$ with $S_{xy}(\tau) = [xy]$ and $S_{xy}(\tau') = [yx]$.

We call a pair of requests $x_{(q)}$, $y_{(l)}$ an *agile pair of* σ if $x_{(q)}$ and $y_{(l)}$ are adjacent in σ and so that σ' obtained by swapping $x_{(q)}$ and $y_{(l)}$ in σ gives $S_{xy}(\sigma') \neq S_{xy}(\sigma)$. $\square$

Clearly, if x^i and y^j are agile, then there exists an agile pair in at least one sequence belonging to $\mathcal{P}(x^i y^j)$.

Lemma 5. If $x_{(q)}$, $y_{(l)}$ is an agile pair of σ, then x and y are adjacent in $S(\sigma)$.

Proof. Let σ' be σ with $x_{(q)}$ and $y_{(l)}$ swapped. Then $S_{xy}(\sigma) \neq S_{xy}(\sigma')$ and $S_{st}(\sigma) = S_{st}(\sigma')$ for all $\{s,t\} \subseteq L$ except $\{x, y\}$. But this is possible only if x and y are adjacent in $S(\sigma)$. $\square$

Definition 6. For every $x^i \in U$ let $R(x^i)$ be the set defined as follows: $x_{(q)} \in R(x^i)$ if and only if there exists $y_{(l)}$ and σ with $\sigma_x = x^i$ such that $x_{(q)}, y_{(l)}$ is an agile pair of σ. $\square$

Lemma 7. Let x^i be unary projection and suppose that x^i forms agile pairs involving at least two distinct items. Then $|R(x^i)| = 1$.

Proof. Obviously, $|R(x^i)| > 0$. Suppose that $|R(x^i)| > 1$; we will show that this leads to a contradiction. Then there are two distinct items y, z and a sequence $\tau \in \mathcal{P}(x^i y^j)$ with an agile pair $x_{(q)}, y_{(l)}$, and similarly $\lambda \in \mathcal{P}(x^i z^k)$ with an agile pair $x_{(q')}, z_{(m)}$ with $q \neq q'$ (if $q = q'$ for all choices of $y_{(l)}$ and $z_{(m)}$, then $|R(x^i)| = 1$). We insert k requests to z into τ, but not between $x_{(q)}$ and $y_{(l)}$, to create a sequence σ with $\sigma_{xy} = \tau$ and $\sigma_{xz} = \lambda$ in which both $x_{(q)}, y_{(l)}$ and $x_{(q')}, z_{(m)}$ are adjacent pairs.

Swap the agile pair $x_{(q)}, y_{(l)}$ in σ to obtain σ' with $\{S_{xy}(\sigma), S_{xy}(\sigma')\} = \{[xy], [yx]\}$. We have $\sigma_{xz} = \sigma'_{xz} = \lambda$. Suppose that $S_{xz}(\sigma) = [zx]$ and $S_{xy}(\sigma) = [xy]$ (and hence $S_{xyz} = [zxy]$), or that $S_{xz}(\sigma) = [xz]$ and $S_{xy}(\sigma) = [yx]$ (and hence $S_{xyz} = [yxz]$), otherwise exchange σ and σ'. Now consider the sequence σ'' obtained from σ by swapping both agile pairs $x_{(q)}, y_{(l)}$ and $x_{(q')}, z_{(m)}$. This reverses the three-element list S_{xyz}, that is, $\{S_{xyz}(\sigma), S_{xyz}(\sigma'')\} = \{[zxy], [yxz]\}$, so that $S_{yz}(\sigma) \neq S_{yz}(\sigma'')$, but $\sigma_{yz} = \sigma''_{yz}$, which contradicts the projectivity of the algorithm with respect to y and z. □

Lemma 8. If $x_{(q)}, y_{(l)}$ is an agile pair in $\lambda \in \mathcal{P}(x^i y^j)$ and $|R(x^i)| = 1$ and $|R(y^j)| = 1$, then the only swap of requests that can change the relative order of x and y in a request sequence in $\mathcal{P}(x^i y^j)$ is swapping $x_{(q)}$ and $y_{(l)}$, and this changes $S_{xy}(\sigma)$ in any such sequence σ where $x_{(q)}$ and $y_{(l)}$ are adjacent.

Proof. Only $x_{(q)}$ and $y_{(l)}$ can be swapped to affect the order of x and y because $|R(x^i)| = |R(y^j)| = 1$. If the lemma does not hold, then there exists a sequence σ in $\mathcal{P}(x^i y^j)$ in which we can swap $x_{(q)}$ and $y_{(l)}$ to obtain σ' with $S_{xy}(\sigma) = S_{xy}(\sigma')$. Then we can obtain any sequence in $\mathcal{P}(\sigma)$ by successively transposing adjacent requests, starting from either σ or σ', without ever swapping $x_{(q)}$ and $y_{(l)}$. Thus, the relative order of x and y would be the same for all request sequences in $\mathcal{P}(x^i y^j)$. But we know that swapping $x_{(q)}$ and $y_{(l)}$ changes $S_{xy}(\lambda)$. This is a contradiction. □

In this and the next section, we consider online list update algorithms that move an item to the front of the list after sufficiently many consecutive requests to that item. This behavior is certainly expected for algorithms with a small competitive ratio. In this section, we show that such algorithms, which we call M-regular, can be characterized in terms of "critical requests". In the next section, we use this characterization to show that such algorithms are at best 1.6-competitive.

Definition 9. For a given integer $M > 0$, a deterministic algorithm is called *M-regular* if for each item x and each request sequence σ, item x is in front of all other items after the sequence σx^M.

A randomized algorithm is called M-regular if it is a discrete probability distribution over deterministic M-regular algorithms. □

The algorithms discussed at the end of the introduction are all 1-regular or 2-regular. A projective algorithm that is not M-regular is `FREQUENCY COUNT`, which maintains the items sorted according to decreasing number of past requests; two items which have been requested equally often are ordered by recency of their last request, like in `MTF`. Hence, after serving the request sequence $x^{M+1}y^M$, item x is still in front of y, which shows that `FREQUENCY COUNT` is not M-regular for any M. Projective algorithms that are not M-regular are characterized in Section 5 below, but such "irregular" behavior must vanish in the long run for any algorithm with a good competitive ratio (see Section 6). Hence, the important projective algorithms are M-regular.

The following theorem asserts the existence of critical requests, essentially the unique element of $R(x^i)$ in Lemma 7, for those unary projections x^i where this lemma applies.

For projectivity, the list items may also be maintained in reverse order, described as case (b) in the following theorem; competitive algorithms do not behave like this, as we will show later.

Theorem 10. Let $\mathtt{A}$ be a deterministic projective algorithm over a set L of list items. Then there exists a function

$$F : U \to \mathbb{N}, \qquad F(x^i) \le i \qquad \text{for all } i$$

so that the following holds:

Let Q be a set of unary projections containing projections to at least three different items. Let all unary projections to different items in Q be pairwise agile. Then one of the following two cases (a) or (b) applies.

(a) For all pairs of unary projections x^i, y^j from Q it holds that if $q = F(x^i)$ and $l = F(y^j)$, then

$$S_{xy}(\sigma) = \begin{cases} [xy] & \text{if } x_{(q)} \text{ is requested after } y_{(l)} \text{ in } \sigma \\ [yx] & \text{if } x_{(q)} \text{ is requested before } y_{(l)} \text{ in } \sigma \end{cases} \tag{8}$$

(b) For all pairs of unary projections x^i, y^j from Q it holds that if $q = F(x^i)$ and $l = F(y^j)$, then

$$S_{xy}(\sigma) = \begin{cases} [xy] & \text{if } x_{(q)} \text{ is requested before } y_{(l)} \text{ in } \sigma \\ [yx] & \text{if } x_{(q)} \text{ is requested after } y_{(l)} \text{ in } \sigma \end{cases} \tag{9}$$

Proof. Since all pairs of unary projections in Q are pairwise agile, we can conclude $|R(x^i)| = 1$ for all $x^i \in Q$ by Lemma 7. This allows us to define $F(x^i) = q$ if $x_{(q)} \in R(x^i)$. From Lemma 8 we can conclude that for every pair x^i, y^j, either (8) or (9) holds.

It remains to prove that either all pairs are operated by (8) or by (9). If this was not the case, then it is not hard to see that one can construct a sequence σ which has a pair of critical requests adjacent to each other in σ (which define an agile pair) without the corresponding items being adjacent in $S(\sigma)$, which contradicts Lemma 5: For example, suppose $F(x^i) = q$, $F(y^j) = r$, and $F(z^k) = s$, consider σ in $\mathcal{P}(x^i y^j z^k)$ so that σ has the three consecutive requests $x_{(q)} z_{(s)} y_{(r)}$, and assume that $S_{xyz}(\sigma) = [xyz]$ because $x_{(q)}$ is requested before $y_{(r)}$ according to (9) and because $y_{(r)}$ is requested after $z_{(s)}$ according to (8); then the critical requests $x_{(q)}$ and $z_{(s)}$ are adjacent in σ but x and z are not adjacent in $S(\sigma)$. □

The following theorem asserts that, in a list with at least three items, an M-regular algorithm operates according to critical requests as in Definition 1 for all pairs of items that have been requested M or more times. That is, case (b) of Theorem 10, where the list items are arranged backwards, does not apply. In addition, the critical request to any item must be one of the last M requests to that item, which means that the relative critical request $f(x^i)$ in (3) is less than M.

Theorem 11. Let A be a deterministic projective M-regular algorithm over a set L of at least three list items. Then there exists a function

$$F : U \to \mathbb{N}, \qquad F(x^i) \leq i \qquad \text{for all } i$$

so that the following holds. Let $x, y \in L$. Let σ be any request sequence with $|\sigma_x| \geq M$ and $|\sigma_y| \geq M$. Then, with $q = F(x^i)$ and $l = F(y^j)$,

$$S_{xy}(\sigma) = \begin{cases} [xy] & \text{if } x_{(q)} \text{ is requested after } y_{(l)} \text{ in } \sigma \\ [yx] & \text{if } x_{(q)} \text{ is requested before } y_{(l)} \text{ in } \sigma \end{cases}$$

Moreover, with $f(x^i)$ defined as in (3), we have $f(x^i) < M$ for all i.

Proof. Let Q be the set of all unary projections x^i with $i \geq M$. This set has all the properties of the set Q in Theorem 10, where clearly case (a) applies because $S_{xy}(x^M y^M) = [yx]$. Because A is M-regular, for $i \geq M$ the critical request $F(x^i)$ is one of the last M requests to x, which shows that $f(x^i) < M$; for $i < M$ this holds trivially. □

4 The Lower Bound for M-regular Algorithms

In this section, we use Theorem 10 to prove the following result.

Theorem 12. No M-regular projective algorithm is better than 1.6-competitive.

We first give an outline of the proof. Given any $\varepsilon > 0$ and b, we will show that there is a discrete probability distribution π on a finite set Λ of request sequences so that

$$\sum_{\lambda \in \Lambda} \pi(\lambda) \frac{\texttt{A}(\lambda)}{\texttt{OPT}(\lambda) + b} \geq 1.6 - \varepsilon, \tag{10}$$

for any deterministic M-regular algorithm A. Then *Yao's theorem* [17] asserts that also any randomized M-regular algorithm has competitive ratio $1.6 - \varepsilon$ or larger. This holds for any $\varepsilon > 0$, so the competitive ratio is at least 1.6. This ratio is achieved by COMB, and therefore 1.6 is a tight bound for the competitive ratio of M-regular algorithms.

All $\lambda \in \Lambda$ will consist only of requests to two items x and y. In what follows, let $\hat{M} \geq M$ and $\hat{M} \geq 3$ and let the request sequence ϕ be

$$\phi := x^{\hat{M}}\, yx^{\hat{M}}\, y^{\hat{M}}\, xy^{\hat{M}}\, x^{\hat{M}}\, yxyx^{\hat{M}}\, y^{\hat{M}}\, xyxy^{\hat{M}}\,. \tag{11}$$

By the last observation in Theorem 11, x will be in front of the list after any subsequence $x^{\hat{M}}$ of requests, and y after any subsequence $y^{\hat{M}}$. The purpose of the following construction is to obscure to the algorithm (which operates according to critical requests defined by the unary projections) the exact location of a request to x or y in a repetition of ϕ.

Let K and T be positive integers and let H be the number of requests to x (and to y) in ϕ, that is,

$$H := |\phi|/2 = 4\hat{M} + 4. \tag{12}$$

Then the set Λ of sequences in (10) is given by

$$\Lambda = \Lambda(K,T) := \{x^{\hat{M}+t}y^{\hat{M}+h}\phi^K \mid 0 \le h < H, 0 \le t < HT\}, \tag{13}$$

where π chooses any λ in Λ with equal probability $\pi(\lambda) = 1/H^2T$. Note that in (13), K is the number of repetitions of ϕ, the number H depends on $\hat{M}$ but is otherwise constant, h creates a prefix for y so as to achieve any possible position inside ϕ for a given request to y, and T is a second parameter that defines the range of t so that the number of requests to x can vary widely relative to y; it is not necessary to introduce such a parameter for y.

It is easy to see that OPT pays exactly ten units for each repetition of ϕ (which always starts in offline list state $[yx]$). Assuming that the initial list state is also $[yx]$, all sequences in Λ have offline cost $10K + 2$. This and the fact that $\pi(\lambda)$ for $\lambda \in \Lambda$ is constant allows us to show (10) once we can prove – which we will do in the course of our argument –

$$\sum_{\lambda\in\Lambda} \mathtt{A}(\lambda) \ge 16KH^2T - o(KH^2T), \tag{14}$$

because then

$$\sum_{\lambda\in\Lambda}\pi(\lambda)\frac{\mathtt{A}(\lambda)}{\mathtt{OPT}(\lambda)+b} = \frac{\sum_{\lambda\in\Lambda}\mathtt{A}(\lambda)}{\sum_{\lambda\in\Lambda}(\mathtt{OPT}(\lambda)+b)} \ge \frac{16KH^2T - o(KH^2T)}{(10K+2+b)H^2T} \ge 1.6 - \varepsilon$$

for K and T large enough.

Recall that by Theorem 11, the algorithm uses critical requests that depend only on the unary projections x^i and y^j to x and y of a sequence in Λ. We refer to the pair (i, j) as a *state*, according to the following definition.

Definition 13. A request sequence σ ends at *state* (i, j) if $|\sigma_x| = i$ and $|\sigma_y| = j$. The request sequence λ *passes* state (i, j) if there is a proper prefix σ of λ, with $\lambda = \sigma\tau$ for non-empty τ, so that σ ends at (i, j). The request in λ *after* (i, j) is the first request in τ. □

Definition 14. Let $\mathtt{A}_\lambda(i, j)$ denote the online cost of serving the requests in λ after (i, j). If λ does not pass (i, j), let $\mathtt{A}_\lambda(i, j) = 0$. □

We will show that the set Λ in (13) is constructed in such a way that almost all states which are passed by some sequence λ in Λ are so-called *good states*, defined as follows.

Definition 15. A state (i, j) is called *good* if for every proper prefix σ of ϕ (that is, $0 \le |\sigma| < 2H$) there exist unique h, k, t with $0 \le h < H$, $0 \le k < K$ and $0 \le t < HT$ so that $x^{\hat{M}+t}y^{\hat{M}+h}\phi^k\sigma$ ends at state (i, j). □

Note that H is the number of requests to y in ϕ, so given (i, j) and the prefix σ of ϕ in Definition 15, there is at most one choice of h and k, and therefore at most one t, so that $x^{\hat{M}+t}y^{\hat{M}+h}\phi^k\sigma$ ends at state (i, j) (see also (16) below). The state (i, j) is good if these h, k, t exist for all proper prefixes σ of ϕ, which means that each position inside the repetition of ϕ in the sequence chosen randomly from Λ is equally likely.

The following Lemma 16 states that good states incur large costs. After that we prove that almost all states are good and thus complete the proof of Theorem 12.

Lemma 16. Let (i, j) be a good state. Then

$$\sum_{\lambda\in\Lambda} \mathtt{A}_\lambda(i, j) \geq 16.$$

Proof. Consider any sequence λ in Λ so that λ passes (i, j); there are $2H$ such sequences by Definition 15. The request in λ after (i, j) is some request in ϕ. The cost $\mathtt{A}_\lambda(i, j)$ of serving that request depends on whether the requested item x or y is in front or not. This, in turn, is determined by the terms $f(x^i)$ and $f(y^j)$ as defined in (3), which determine the relative critical requests to x and y in λ. Recall that the item with the more recent critical request is in front, and that $f(x^i)$ and $f(y^j)$ are less than $\hat{M}$ by Theorem 11.

Because (i, j) is a good state, we obtain exactly all the requests in ϕ as the requests after (i, j) in λ when considering all λ in Λ that pass (i, j). Therefore, the total cost $\sum_{\lambda\in\Lambda} \mathtt{A}_\lambda(i, j)$ is the cost of serving exactly the requests in ϕ according to the critical requests as given by $f(x^i)$ and $f(y^j)$.

$f(x^i)$	$f(y^j)$	$x^{\hat{M}}$	$yx^{\hat{M}}$	$y^{\hat{M}}$	$xy^{\hat{M}}$	$x^{\hat{M}}$	$yxyx^{\hat{M}}$	$y^{\hat{M}}$	$xyxy^{\hat{M}}$	$\sum_{\lambda\in\Lambda} \mathtt{A}_\lambda(i, j)$
0	0	1..	11..	1..	11..	1..	1111..	1..	1111..	16
0	≥1	1..	1..	11..	111..	1..	101..	11..	11011..	≥16
1	1	11..	1..	11..	1..	11..	1011..	11..	1011..	16
1	≥2	11..	1..	111..	1..	11..	101..	111..	10111..	≥18
≥2	≥2	111..	1..	111..	1..	111..	101..	111..	101..	≥18

Table 1: Online costs $\mathtt{A}_\lambda(i, j)$ for all λ that pass a good state (i, j), which are the costs of serving the requests in ϕ. They depend on the relative critical requests $f(x^i)$ and $f(y^j)$.

The rows in Table 1 show the costs $\mathtt{A}_\lambda(i, j)$ for the possible combinations of $f(x^i)$ and $f(y^j)$, up to symmetry in x and y (explained further at the end of this proof). For example, consider the first case $f(x^i) = 0$ and $f(y^j) = 0$, where the critical request to an item is always the most recent request to that item, which is the `MTF` algorithm. Suppose that the request after (i, j) is the first request, to x, in the subsequence $xy^{\hat{M}}$ of ϕ. The critical request to x is the last request to x earlier in $yx^{\hat{M}}$, and the critical request to y is the last request to y earlier (and more recent) in $y^{\hat{M}}$. The critical request to y is later than that to x, so y is in front of x, and serving x incurs cost 1, which is the first 1 in the table entry $11..$ in the column for $xy^{\hat{M}}$. The second 1 in $11..$ is the cost of serving the first y. It is 1 because here the critical request to x is more recent than the critical request to y. The ".." in $11..$ correspond to the costs of later requests to y in $y^{\hat{M}}$, which are zero for $f(x^i) = 0$ and $f(y^j) = 0$ (so for $\hat{M} = 4$ the complete cost sequence would be 11000). In a good state, each cost 0 or 1 in the table (in correspondence to the respective position in ϕ) is incurred by a sequence λ in Λ.

By construction of Λ, the requests before $x^{\hat{M}}$ in the first column of Table 1 are of the form $y^{\hat{M}}$, so y is in front of x, and the first request of $x^{\hat{M}}$ has always cost 1.

In the second row in Table 1, $f(x^i) = 0$ and $f(y^j) \geq 1$; if $f(y^j) = 1$, then the request to y is handled as in the `TS` algorithm. As an illustration of a more complicated case, consider the subsequence $xyxy^{\hat{M}}$ of ϕ in the last column, with associated costs $11011...$ The first 1 is the cost of serving the first request to x, because the preceding requests are $\hat{M} \geq M$ requests to y in $y^{\hat{M}}$ and because the algorithm is M-regular, which means $f(y^j) < M$ by Theorem 11, so y is in front of x. Because $f(x^i) = 0$, the cost of serving the first y in $xyxy^{\hat{M}}$ is also 1, because x is in front of y. The second request to x has cost 0 (the first 0 in $11011..$) because y is not moved in front of x (the critical request to y is earlier than that to x because $f(y^j) \geq 1$). The next two costs 11 are for the second and third request to y in $xyxy^{\hat{M}}$, because the critical request to x is more recent.

The rows in Table 1 describe all cases for $f(x^i)$ and $f(y^j)$ with $i \leq j$. They describe in fact all possible cases because for each column in Table 1 there is another column with x and y interchanged, where the costs for requests to x and y apply in the same manner when x is exchanged with y. The respective costs in Table 1 are easily verified. The right column shows that the total cost $\sum_{\lambda\in\Lambda} \mathtt{A}_\lambda(i,j)$ is at least 16 in all these cases, which proves the claim. □

The preceding proof of Lemma 16 also shows that 1.6-competitive algorithms can only be expected when the relative critical requests fulfill $f(x^i) \in \{0, 1\}$, as in the `MTF` and `TS` algorithms.

Proof of Theorem 12. We only have to prove (14), which we will do by showing

$$\sum_{\lambda\in\Lambda} \mathtt{A}(\lambda) \geq \sum_{(i,j)\text{ good}} \sum_{\lambda\in\Lambda} \mathtt{A}_\lambda(i,j) \geq 16KH^2T - o(KH^2T). \tag{15}$$

The first inequality in (15) is immediate. For the second inequality we use Lemma 16. It suffices to show that the number of good states is at least

$$KH^2T - o(KH^2T).$$

By Definition 15, state (i, j) is good if and only if

$$\begin{aligned} i &= \hat{M} + t + kH + |\sigma_x|, \\ j &= \hat{M} + h + kH + |\sigma_y|, \end{aligned} \tag{16}$$

or equivalently

$$\begin{aligned} t &= i+h-j-(|\sigma_x|-|\sigma_y|), \\ h+kH &= j-\hat{M}-|\sigma_y|. \end{aligned} \tag{17}$$

For $0 \le k < K$ and $0 \le h < H$, the term $h + kH$ takes the values $0, \ldots, KH - 1$. The second equation in (17) therefore has a unique solution in h, k, for any σ (where $0 \le |\sigma_y| < H$) whenever $\hat{M} + H - 1 \le j < \hat{M} + KH$. Because by (11), $0 \le |\sigma_x| - |\sigma_y| < H$, the first equation in (17) has a unique solution t in $\{0, \ldots, HT - 1\}$ if $j + H - 1 \le i \le j + HT - H$, for every fixed j. Hence the number of good states is at least

$$(KH - H + 1) \cdot (HT - 2H + 2) = KH^2T - o(KH^2T)$$

because for sufficiently large K (the number of repetitions of ϕ) and T (the number of initial repetitions of x) all other terms are arbitrarily small relative to KH^2T. □

5 The Full Characterization

In this section, we give the full characterization of deterministic projective algorithms. We consider the set U of unary projections of request sequences defined in (2) as the set of nodes of the directed graph $G = (U, E)$ with arcs (x^i, y^j) in E whenever there is a request sequence σ in $\mathcal{P}(x^i y^j)$ with $S(\sigma) = [xy]$.

For any two distinct items x and y and any $i, j \ge 0$, there is at least one arc between x^i and y^j. If the pair x^i, y^j is agile according to Definition 4, then there are arcs in both directions. Only pairs of nodes of the form x^i, x^j do not have arcs between them.

Let $\mathcal{W}$ be the set of strongly connected components of G, and let $C(x^i)$ be the strongly connected component that x^i belongs to. We think of $C(x^i)$ as a "container" that contains x^i and all other unary projections y^j with $C(y^j) = C(x^i)$.

There exists a total order $<$ on these containers so that $C(x^i) < C(y^j)$ if $S_{xy}(\sigma) = [xy]$ after serving any $\sigma \in \mathcal{P}(x^i y^j)$. To see this, we define the following binary relation P on $\mathcal{W}$: Let $C(x^i)\, P\, C(y^j)$ if there is a path in G from x^i to y^j. Then P defines a partial order on $\mathcal{W}$. It is acyclic because cycles in G belong to strongly connected components, which are the elements of $\mathcal{W}$. The only pairs of containers which are not ordered in P are those of the form $\{x^i\}$, $\{x^j\}$ for which there does not exist a container $C(y^k)$ with $C(x^i) < C(y^k) < C(x^j)$ or $C(x^j) < C(y^k) < C(x^i)$. By stipulating $\{x^i\} < \{x^j\}$ if and only if $i < j$ for such pairs, we can extend P to the desired total order $<$.

A specific case is given by the empty unary projections x^0 for items x: Note that x^0 and y^j for any $j \ge 0$ are never in the same container because $\mathcal{P}(x^0 y^j)$ contains only a single sequence $\sigma = y^j$; the state $S_{xy}(\sigma)$ is therefore either $[xy]$ or $[yx]$, so there cannot be paths in both directions between x^0 and y^j in G. Hence $C(x^0) = \{x^0\}$, and $C(x^0) < C(y^0)$ if and only if x is in front of y in the initial list.

In summary, for a request sequence σ, the total order $<$ on $\mathcal{W}$ determines the list order between two items x and y whose unary projections σ_x and σ_y belong to different containers in $\mathcal{W}$.

If σ_x and σ_y belong to the same container, then the list order between x and y can be described by essentially two possibilities. First, if the container contains only projections to at most two items x and y, nothing further can be said because the relative order between x and y for these requests is arbitrary without violating projectivity (for the same reason that on a two-item list, any algorithm is projective); the set of these containers will be denoted by $\mathcal{W}_2$.

Second, if a container contains unary projections for three or more distinct items, then the algorithm's behavior can be described by critical requests similar to Theorem 10; the set of such containers will be denoted by $\mathcal{W}^+$. There is a symmetric set $\mathcal{W}^-$ where the algorithm behaves in the same manner but with the list order reversed (which does not define competitive algorithms).

These assertions are summarized in the following theorem.

Theorem 17. Consider a deterministic projective list update algorithm. Then there are pairwise disjoint sets $\mathcal{W}^+$, $\mathcal{W}^-$, $\mathcal{W}_2$ whose union is $\mathcal{W}$ and a total order $<$ on $\mathcal{W}$ and a function $C : U \to \mathcal{W}$ with

(I) $C(x^0) = \{x^0\} \in \mathcal{W}_2$ for all $x \in L$;

(II) for any three items x, y, z, if $C(x^i) = C(y^j) = C(z^k) = w$, then $w \notin \mathcal{W}_2$.

Furthermore, if $C(x^i) \notin \mathcal{W}_2$, then there exists $F(x^i) \in \{1, \dots, i\}$ with the following properties: For all request sequences σ with $\sigma_x = x^i$ and $\sigma_y = y^j$,

(III) if $C(x^i) < C(y^j)$ then $S_{xy}(\sigma) = [xy]$;

(IVa) if $C(x^i) = C(y^j) \in \mathcal{W}^+$ then $S_{xy}(\sigma) = [xy]$ if and only if the $F(x^i)$th request to x is *after* the $F(y^j)$th request to y in σ;

(IVb) if $C(x^i) = C(y^j) \in \mathcal{W}^-$ then $S_{xy}(\sigma) = [xy]$ if and only if the $F(x^i)$th request to x is *before* the $F(y^j)$th request to y in σ.

Proof. The set $\mathcal{W}$ and the order $<$ have been defined above with the help of the graph G, which shows (III). We have also shown (I) above.

As before, let $\mathcal{W}_2$ be the set of containers with unary projections to at most two distinct items, which implies (II).

It remains to show (IVa) and (IVb). Consider a request sequence σ with $\sigma_x = x^i$ and $\sigma_y = y^j$. Let $C(x^i) = C(y^j) \notin \mathcal{W}_2$, so that there is a third item $z \notin \{x, y\}$ with $C(x^i) = C(y^j) = C(z^k)$. We want to apply Lemma 7. To this end, we first show the "mixed transitivity" (note that x, y, z are distinct items)

$$(x^i, y^j) \in E \quad \text{and} \quad (y^j, z^k) \in E \quad \Longrightarrow \quad (x^i, z^k) \in E. \tag{18}$$

Let $(x^i, y^j) \in E$, so that $S_{xy}(\sigma) = [xy]$ for some $\sigma \in \mathcal{P}(x^i y^j)$. If $(y^j, z^k) \in E$, then one can insert k requests to z into σ so that $S_{yz}(\sigma) = [yz]$. Adding the requests to z does not change $S_{xy}(\sigma)$, so $S(\sigma) = [xyz]$, which implies $(x^i, z^k) \in E$. This shows (18).

With the help of (18), we now show that if $C(x^i) = C(y^j)$, then the pair x^i, y^j is agile according to Definition 4. We will prove this by showing that

$$(x^i, y^j) \in E \quad \text{and} \quad (y^j, x^i) \in E. \tag{19}$$

To prove (19), recall that $C(x^i)$ is a strongly connected component of the graph G which also contains y^j and z^k. Therefore there exists a path in G from x^i to y^j via z^k. This path is a sequence of unary projections $u_0, \ldots, u_n$ with $u_0 = x^i$, $u_l = z^k$ for some $0 < l < n$, and $u_n = y^j$. Let s_i be the item of the corresponding unary projection u_i, in particular $s_0 = x$, $s_l = z$, $s_n = y$. Ignoring the superscripts of the unary projections, we are essentially looking at a sequence of items $s_0 s_1 s_2 \ldots s_n$ where $s_i \neq s_{i+1}$ for $0 \leq i < n$. We can shorten that sequence whenever s_{q-1}, s_q, and s_{q+1} are three distinct items by removing s_q, because then $(s_{q-1}, s_{q+1}) \in E$ by (18). The problem is that we do not want to shorten it in such a way that we cannot apply (18) any more.

We call a path $u_0 \ldots u_n$ between x^i and y^j *valid* if $|\{s_0, \ldots, s_n\}| \geq 3$. We claim that if there exists a valid path between x^i and y^j of length $n > 2$, then there exists also a valid path of length $n - 1$.

To show this claim, consider the smallest q so that s_{q-1}, s_q, and s_{q+1} are three distinct items. If the path $u_0 \ldots u_n$ remains valid after removing u_q, we are done. Otherwise, clearly $|\{s_0, \ldots, s_n\}| = 3$, and removing u_q makes the path invalid, which means $s_q = z$ for some $z \notin \{x, y\}$, and s_q is the only occurrence of z in $s_0 s_1 s_2 \ldots s_n$, because $s_0 = x$ and $s_n = y$. We claim that $q = 1$, because if $q > 1$ then the sequence $s_0 s_1 \ldots s_q$ is either of the form $xyxy \ldots xyz$ or $xyx \ldots yxz$, and in both cases s_{q-1} can be removed, but q was chosen smallest. So indeed $s_1 = z$, and this is the only occurrence of z. Then $s_0 s_1 s_2 \ldots s_n$ is either of the form $xzxy \ldots xy$ or $xzyx \ldots y$, and in each case we can repeatedly remove s_2 using (18), until we arrive at $n = 2$ with the sequence xzy. This proves the claim.

A final application of (18) then gives $(x^i, y^j) \in E$. The same argument shows $(y^j, x^i) \in E$. This proves (19).

Because all pairs of unary projections are agile in $C(x^i)$, we can apply Theorem 10, whose cases (a) and (b) prove (IVa) and (IVb). This proves the theorem. □

6 The Lower Bound for Irregular Algorithms

In Section 4 we considered deterministic M-regular projective list update algorithms. In this section, we consider randomized algorithms, which may select deterministic algorithms that are not M-regular. If this happens sufficiently rarely, the algorithm may still be competitive. For example, consider an algorithm that operates according to some rule (for example `MTF`), keeps track of its incurred costs, and whenever this is a square number Q^2, does not move any item for the next Q requests, and then resumes its normal operation. This does not change its competitive ratio, but makes the algorithm no longer M-regular.

In Theorem 12 we showed that no deterministic M-regular projective list update algorithm is better than 1.6-competitive. For this we gave, for any $\varepsilon > 0$, a suitable distribution on request sequences that bound the competitive ratio of the algorithm from below by $1.6 - \varepsilon$. These request sequences are drawn from a set Λ defined in (13) with parameters K, T that are chosen sufficiently large depending on ε.

We extend this analysis to arbitrary randomized projective list update algorithms using the full characterization from the previous section. Part of this extension involves also a sufficiently large choice of the parameter $\hat{M}$ in (13) to cope with algorithms that are not $\hat{M}$-regular.

In brief, the proof works as follows. Using the crucial notion of a good state (i, j) in Definition 15, we call a deterministic algorithm $\hat{M}$-regular *in state* (i, j) if it fulfills a certain condition, (20) below, where the algorithm only uses the containers from Theorem 17 in the normal way that one expects from competitive algorithms. The lower bound from Lemma 16 applies in expectation for algorithms that fulfill condition (20).

The proof of the following theorem is mostly concerned with the cases where the deterministic algorithm A is "irregular", that is, condition (20) fails. Here we use the following argument, spelled out in detail following (25): We give simple request sequences (which depend on the growing parameters $K, T, \hat{M}$) that have constant offline cost but arbitrarily large cost for deterministic "irregular" algorithms; hence, these deterministic algorithms must be chosen with vanishing probability.

Theorem 18. Any randomized projective list update algorithm that accesses a list of at least three items is at best 1.6-competitive.

Proof. Assume the list has at least three items. Consider a randomized projective algorithm $\mathcal{A}$ and assume that $\mathcal{A}$ is c-competitive with $c < 1.6$. That is, there exists a constant b such that $\mathcal{A}(\sigma) \leq c \cdot \mathtt{OPT}(\sigma) + b$ for all request sequences σ.

We adapt the proof for M-regular algorithms of Section 4. Let $\hat{M} \geq 3$, consider Λ in (13) and consider a good state (i, j) as defined in Definition 15.

Let A be a deterministic projective algorithm. We say that algorithm A is *$\hat{M}$-regular in state* (i, j) if, with $\mathcal{W}^+$ as in Theorem 17 and $f(x^i)$ defined as in (3),

$$C(x^i) = C(y^j) \in \mathcal{W}^+, \qquad f(x^i) < \hat{M}, \qquad f(y^j) < \hat{M}. \tag{20}$$

It is easy to see that the proof of Lemma 16 applies if A is $\hat{M}$-regular in (i, j).

Recall that $\mathcal{A}$ is just a discrete probability distribution on the set of deterministic projective algorithms. Let r_{ij} be the event that $\mathcal{A}$ is $\hat{M}$-regular in state (i, j). Analogous to (15), the expected cost of $\mathcal{A}$ is bounded by considering the good states (i, j) as follows:

$$\begin{aligned} E\left[\sum_{\lambda\in\Lambda} \mathcal{A}(\lambda)\right] &\geq \sum_{(i,j)\text{ good}} E\left[\sum_{\lambda\in\Lambda} \mathcal{A}_\lambda(i,j)\right] \\ &\geq 16KH^2T - o(KH^2T) - \sum_{(i,j)\text{ good}} 16(1-\mathrm{prob}(r_{ij})). \end{aligned} \tag{21}$$

Let

$$X := HT + KH + \hat{M} \quad \text{and} \quad Y := (K+1)H + \hat{M}, \tag{22}$$

and recall that H is a linear function of $\hat{M}$ by (12). For all good states (i,j) we have, by (16),

$$1 \le i \le X \text{ and } 1 \le j \le Y. \tag{23}$$

If we can prove that, with growing $\hat{M}$, K, and T,

$$\sum_{(i,j) \text{ good}} 16(1-\mathrm{prob}(r_{ij})) \le \sum_{j=1}^{Y}\sum_{i=1}^{X} 16(1-\mathrm{prob}(r_{ij})) = o(KH^2T), \tag{24}$$

then we have proved (14) for irregular algorithms.

We proceed to prove (24) by analyzing where (20) fails, that is, for each of the six cases according to

$$\sum_{j=1}^{Y}\sum_{i=1}^{X}(1-\mathrm{prob}(r_{ij})) \le \sum_{j=1}^{Y}\sum_{i=1}^{X}\left(\begin{array}{l} \quad\mathrm{prob}(C(x^i) < C(y^j)) \\ +\,\mathrm{prob}(C(x^i) > C(y^j)) \\ +\,\mathrm{prob}(C(x^i) = C(y^j) \in \mathcal{W}^-) \\ +\,\mathrm{prob}(C(x^i) = C(y^j) \in \mathcal{W}_2) \\ +\,\mathrm{prob}(C(x^i) = C(y^j) \in \mathcal{W}^+,\ f(x^i) \ge \hat{M}) \\ +\,\mathrm{prob}(C(x^i) = C(y^j) \in \mathcal{W}^+,\ f(y^j) \ge \hat{M}) \end{array}\right).$$

We start by proving

$$\sum_{j=1}^{Y}\sum_{i=1}^{X}\mathrm{prob}(C(x^i) < C(y^j)) \le o(KH^2T). \tag{25}$$

To this aim, consider the sequence $x^i y^Y$ for $1 \le i \le X$. When serving this sequence, a request to y will be served in each of the states $(i,1),\ldots,(i,j),\ldots,(i,Y)$. Since every deterministic algorithm with $C(x^i) < C(y^j)$ pays one unit for accessing y in state (i,j), the expected cost of $\mathcal{A}$ for serving a request to y in a state (i,j) is at least $\mathrm{prob}(C(x^i) < C(y^j))$. Therefore

$$\mathcal{A}(x^i y^Y) \ge \sum_{j=1}^{Y}\mathrm{prob}(C(x^i) < C(y^j)). \tag{26}$$

On the other hand, $\mathcal{A}(x^i y^Y) \le c \cdot \mathtt{OPT}(x^i y^Y) + b$ because $\mathcal{A}$ is c-competitive. Since $\mathtt{OPT}(x^i y^Y) = 1$ (the initial list state is $[xy]$) it follows that

$$\sum_{i=1}^{X}\sum_{j=1}^{Y}\mathrm{prob}\left(C(x^i) < C(y^j)\right) \le \sum_{i=1}^{X}\mathcal{A}(x^i y^Y) \le X\cdot(c+b) = o(KH^2T) \tag{27}$$

as desired.

The bound on $\mathrm{prob}(C(x^i) > C(y^j))$ is very similar, using request sequences of the form $y^j x^X$ for $1 \le j \le Y$.

For $\text{prob}(C(x^i) = C(y^j) \in \mathcal{W}^-)$, we use, like for (25), request sequences of the form $\sigma = x^i y^Y$. Clearly, from the first request to y onwards, the critical request to x is always earlier in σ than the critical request to y. Therefore $C(x^i) = C(y^j) \in \mathcal{W}^-$ implies that y is behind x in the list, so

$$\mathcal{A}(x^i y^Y) \geq \sum_{j=1}^{Y} \text{prob}(C(x^i) = C(y^j) \in \mathcal{W}^-),$$

and the same argument as after (26) applies.

If $C(x^i) = C(y^j) \in \mathcal{W}_2$, the container $C(x^i)$ does not contain any unary projections to items other than x or y. The list has at least a third item z and either $C(x^i) < C(z^k)$ or $C(z^k) < C(x^i)$ for any k. We consider only the first case, where we can bound $\text{prob}(C(x^i) < C(z^k))$ similarly to (25). By considering the request sequence $x^i z^Y$ for $1 \leq i \leq X$, we obtain in the same way as with (26) and (27) that $\sum_{i=1}^{X} \mathcal{A}(x^i z^Y) = o(KH^2T)$.

As explained, if $C(x^i)$ and $C(y^j)$ are two containers in $\mathcal{W}_2$, then either $C(x^i) < C(z^k)$ or $C(x^i) > C(z^k)$ for all z^k with $z \neq x, y$, so that

$$\text{prob}(C(x^i) = C(y^j) \in \mathcal{W}_2) \leq \text{prob}(C(x^i) < C(z^k)) + \text{prob}(C(x^i) > C(z^k)).$$

Hence the left hand side can be bounded by the bound on the first two cases.

In a similar fashion, we bound $\text{prob}(C(x^i) = C(y^j) \in \mathcal{W}^+,\ f(x^i) \geq \hat{M})$. First of all, $C(x^i) = C(y^j) \in \mathcal{W}^+$ implies that both x^i and y^j are in the same container and have critical requests, that is, $F(x^i) > 0$ and $F(y^j) > 0$, so the relative requests in (3) fulfill $f(x^i) < i$ and $f(y^j) < j$. So $f(x^i) \geq \hat{M}$ implies $i > M$, and therefore

$$\begin{aligned} &\sum_{j=1}^{Y} \sum_{i=1}^{X} \text{prob}(C(x^i) = C(y^j) \in \mathcal{W}^+,\ f(x^i) \geq \hat{M}) \\ = &\sum_{j=1}^{Y} \sum_{i=\hat{M}+1}^{X} \text{prob}(C(x^i) = C(y^j) \in \mathcal{W}^+,\ f(x^i) \geq \hat{M}). \end{aligned} \tag{28}$$

Next, we show

$$\begin{aligned} &\sum_{i=\hat{M}+1}^{X} \text{prob}(C(x^i) = C(y^j) \in \mathcal{W}^+,\ f(x^i) \geq \hat{M}) \\ \leq\ &\frac{1}{\hat{M}} \sum_{i'=1}^{X} \sum_{\ell=1}^{\hat{M}} \text{prob}(C(x^{i'+\ell}) = C(y^j) \in \mathcal{W}^+,\ f(x^{i'+\ell}) \geq \hat{M}). \end{aligned} \tag{29}$$

Namely, for each $i = \hat{M}+1, \ldots, X$ there are at least $\hat{M}$ choices i', ℓ so that $i = i' + \ell$, because for each $\ell = 1, \ldots, \hat{M}$ the term $i' = i - \ell$ fulfills $1 \leq i' \leq X$. This shows (29).

Let $1 \le j \le Y$ and $1 \le i' \le X$ and consider in the request sequence $x^{i'} y^j x^{\hat{M}}$ (which has constant offline cost) the last $\hat{M}$ requests to x. If their critical request is before the critical request to y (which exists), they incur online cost one, so

$$\begin{aligned} & \sum_{\ell=1}^{\hat{M}} \mathrm{prob}(C(x^{i'+\ell}) = C(y^j) \in \mathcal{W}^+,\ f(x^{i'+\ell}) \ge \hat{M}) \\ \le\ & \mathcal{A}(x^{i'} y^j x^{\hat{M}}) \le c \cdot \mathtt{OPT}(x^{i'} y^j x^{\hat{M}}) + b = O(1). \end{aligned} \tag{30}$$

Consider (28), (29), and (30) and note that $Y/\hat{M} = O(K)$ by (22) and (12), so $Y/\hat{M} \cdot X = O(KHT + K^2 H)$. This shows

$$\sum_{j=1}^{Y} \sum_{i=1}^{X} \mathrm{prob}(C(x^i) = C(y^j) \in \mathcal{W}^+,\ f(x^i) \ge \hat{M}) \le Y \frac{1}{\hat{M}} X \cdot O(1) = o(KH^2 T)$$

if we let $K, T, \hat{M}$ (and thus H) grow while keeping K/T constant.

The bound on $\mathrm{prob}(C(x^i) = C(y^j) \in \mathcal{W}^+,\ f(y^j) \ge \hat{M})$ is proved analogously to the previous bound. □

7 Conclusions

An open problem is to extend the lower bound 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 [7] requires new insights.

Finally, the search for good non-projective algorithms has become an issue with our result. Irani's SPLIT algorithm [13] 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.

A first result is a non-projective algorithm for lists of up to four items based on partial orders which is 1.5-competitive [3]; for another study of algorithms for short lists see [12]. Extending the partial order approach to longer lists is not straightforward (and has in fact led to the lower bounds of 1.501 for lists of length five in [7] and [5, p. 38]).

References

[1] S. Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27(3):682–693, 1998.

[2] S. Albers, B. von Stengel, and R. Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Information Processing Letters*, 56(3):135–139, 1995.

[3] S. Albers, B. von Stengel, and R. Werchner. List update posets. Unpublished, available at `http://www.maths.lse.ac.uk/Personal/stengel/TEXTE/listupdateposets.pdf`, 1996.

[4] S. Albers and J. Westbrook. Self-organizing data structures. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms*, volume 1442 of *Lecture Notes in Computer Science*, pages 13–51. Springer, 1998.

[5] C. Ambühl. *On the List Update Problem*. PhD thesis, ETH Zürich, 2002.

[6] C. Ambühl, B. Gärtner, and B. von Stengel. Optimal projective algorithms for the list update problem. In *Proceedings of 27th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 305–316, 2000.

[7] C. Ambühl, B. Gärtner, and B. von Stengel. A new lower bound for the list update problem in the partial cost model. *Theoretical Computer Science*, 268(1):3–16, 2001.

[8] J. L. Bentley and C. C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. *Communications of the ACM*, 28(4):404–411, 1985.

[9] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, New York, NY, USA, 1998.

[10] R. Dorrigiv, M. R. Ehmsen, and A. López-Ortiz. Parameterized analysis of paging and list update algorithms. In E. Bampis and K. Jansen, editors, *WAOA 2009*, volume 5893 of *Lecture Notes in Computer Science*, pages 104–115. Springer, 2010.

[11] M. R. Ehmsen, J. S. Kohrt, and K. S. Larsen. List factoring and relative worst order analysis. In K. Jansen and R. Solis-Oba, editors, *WAOA 2010*, volume 6534 of *Lecture Notes in Computer Science*, pages 118–129. Springer, 2011.

[12] T. Hagerup. Online and offline access to short lists. In L. Kucera and A. Kucera, editors, *MFCS*, volume 4708 of *Lecture Notes in Computer Science*, pages 691–702. Springer, 2007.

[13] S. Irani. Two results on the list update problem. *Information Processing Letters*, 38(6):301–306, 1991. Corrected version appeared as Technical Report 96-53, ICS Department, U.C. Irvine, CA, USA 1996.

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

[15] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[16] B. Teia. A lower bound for randomized list update algorithms. *Information Processing Letters*, 47(1):5–9, 1993.

[17] A. C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *Proceedings of 19th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 222–227, 1977.