INSTYTUT INFORMATYKI
UNIWERSYTETU WROCŁAWSKIEGO

INSTITUTE OF COMPUTER SCIENCE
UNIVERSITY OF WROCŁAW

ul. Joliot-Curie 15
50–383 Wrocław
Poland

Report 02/08

Marcin Bienkowski, Marek Chrobak, Christoph Dürr, Mathilde Hurand, Artur Jeż, Łukasz Jeż, Jakub Łopuszański and Grzegorz Stachowiak

Generalized Whac-a-Mole

March 2008

Generalized Whac-a-Mole

Marcin Bienkowski[1], Marek Chrobak[2], Christoph Dürr[3], Mathilde Hurand[3], Artur Jeż[1], Łukasz Jeż[1], Jakub Łopuszański[1], and Grzegorz Stachowiak[1]

[1] Institute of Computer Science, University of Wrocław, 50-383 Wrocław, Poland. *
[2] Department of Computer Science, University of California, Riverside, CA 92521, USA. **
[3] CNRS, LIX UMR 7161, Ecole Polytechnique 91128 Palaiseau, France. ***

Abstract. We consider online competitive algorithms for the problem of collecting weighted items from a dynamic set $\mathcal{S}$, when items are added to or deleted from $\mathcal{S}$ over time. The objective is to maximize the total weight of collected items. We study the general version, as well as variants with various restrictions, including the following: the *uniform case*, when all items have the same weight, the *decremental sets*, when all items are present at the beginning and only deletion operations are allowed, and *dynamic queues*, where the dynamic set is ordered and only its prefixes can be deleted (with no restriction on insertions). The dynamic queue case is a generalization of bounded-delay packet scheduling (also referred to as buffer management).

We present several upper and lower bounds on the competitive ratio for this problem, including the following. For the general version, a simple greedy algorithm is 2-competitive and no better ratio is possible for deterministic algorithms. This lower bound extends, in fact, to randomized algorithms against an adaptive adversary. For the uniform case and decremental sets we give an $\mathrm{e}/(\mathrm{e}-1)$-competitive randomized algorithm (against an oblivious adversary) and prove a matching lower bound.

Most of our results concern dynamic queues. For decremental queues, we present a 1.737-competitive deterministic algorithm and prove a lower bound of ≈ 1.63. For the FIFO case, when insertions are allowed only at the end (this generalizes scheduling packets with agreeable deadlines), we give an algorithm with ratio 1.8. For general dynamic queues, we give a deterministic algorithm that has competitive ratio ϕ for instances with non-decreasing weights. In the randomized case, we show that no memoryless algorithm can have ratio smaller than $\mathrm{e}/(\mathrm{e}-1)$ against an adaptive adversary, matching a known upper bound.

1 Introduction

Whac-a-mole is an old arcade game, where plastic "moles" pop out of holes in the machine for short periods of time, in some unpredictable way, and the player

* Supported by MNiSW grants number N206 001 31/0436, 2006–2008 and N N206 1723 33, 2007–2010.
** Supported by NSF grants OISE-0340752 and CCF-0729071.
*** Supported by ANR Alpage.

uses a mallet to "whack" as many moles as possible. In the generalized version that we consider, multiple moles may be present at the same time, and different moles may have different values.

In a more formal setting, we think of it as a dynamic set $\mathcal{S}$ of weighted items (moles). Before each step, some items can be deleted from $\mathcal{S}$ and other items can be added to $\mathcal{S}$. We are allowed to collect one item (whack one mole) from $\mathcal{S}$ per step. Each item can be collected only once. The objective is to maximize the total weight of the collected items.

To our knowledge, this simple and fundamental problem has not been explicitly addressed in the literature on competitive analysis. By placing appropriate assumptions on the structure of $\mathcal{S}$ or on the type of allowed operations, one can obtain a number of natural and interesting special cases, some of which are closely related to known online problems and our work has some bearing on those problems as well.

We study the general case of dynamic sets, as well as some restricted cases, among which we focus on the following versions:

Dynamic Queue: In this case, $\mathcal{S}$ represents a list, that is the items in $\mathcal{S}$ are ordered. Items can be added to $\mathcal{S}$ at any place, but only a prefix of $\mathcal{S}$ can be deleted. A queue is called a *FIFO queue* if insertions are allowed only at the end.

Decremental Sets: Here, all items are added at the beginning, and only deletions are allowed afterwards. In particular, one can consider the case of decremental queues, where only prefix-deletion operations are allowed.

The case of dynamic queues generalizes the well-studied problem of bounded-delay packet scheduling (also known as buffer management), or, equivalently, the problem of scheduling unit jobs with deadlines for maximum weighted throughput. In this problem, packets with values and deadlines arrive in a buffer of a network link. At each step, we can send one packet along the link. The objective is to maximize the total value of packets sent before their deadlines. This is a special case of our dynamic queue problem, where packets are represented by items ordered according to deadlines. The difference is quite crucial though: in packet scheduling, packet arrival times are unknown but their departure times are known, while in dynamic queues *both* the arrival and departure times are not known. The FIFO case generalizes the version of packet scheduling with agreeable deadlines.

Competitive algorithms for various versions of bounded-delay packet scheduling problem have been extensively studied [1, 4, 6, 7, 11–16]. In particular, it is known that no deterministic online algorithm can have competitive ratio better than $\phi \approx 1.618$ [1, 7], and algorithms with competitive ratio ≈ 1.83 have been recently developed [17, 8]. These are the best lower and upper bounds for this problem and closing this gap remains a tantalizing open problem. For agreeable deadlines, an upper bound of ϕ has been established in [16].

The dynamic set problem has some indirect connections to several other packet scheduling problems in QoS networks, where the objective is to maximize the value of packets that reach their destination under various scenarios [2, 3, 12–15].

A metric variant of the whac-a-mole problem appeared in [10]. In their version moving the whack to a mole takes time and mole dissapearance times are known in advance. In particular, their results are inapplicable in our model.

Our results. We first consider the general case of dynamic sets. For deterministic algorithms, it is quite easy to show a lower bound of 2 (even for two items in the decremental case), and this ratio can be achieved by a simple greedy algorithm that always collects the heaviest item.

For randomized algorithms, we focus on the *uniform case*, when all items have weight 1. We show that no memoryless randomized algorithm can achieve competitive ratio better than 2. We then study the uniform decremental case, for which we give an online randomized algorithm with competitive ratio $e/(e-1)$ (against an oblivious adversary) and prove a matching lower bound.

Most of our results concern dynamic queues. In the deterministic case, even for decremental queues, it is quite easy to show a lower bound of ϕ. We improve this bound, by proving a lower bound of ≈ 1.63, that applies even to the decremental case. We also show that no memoryless algorithm can have a ratio smaller than 2. This contrasts with the result of Englert and Westerman [8] who gave a memoryless algorithm for packet scheduling with ratio ≈ 1.893. Thus, at least for memoryless algorithms, knowing the exact deadlines helps.

As for upper bounds, we give the following deterministic algorithms: (i) A 1.737-competitive algorithm for decremental queues (beating the ratio 2 of the naïve greedy algorithm). (ii) A 1.8-competitive algorithm for FIFO queues. (iii) A ϕ-competitive algorithm for dynamic queues when the item weights are non-decreasing (w.r.t. their position in the queue). This last result has some useful implications for packet scheduling, as all lower bound proofs for that problem use instances where packets' weights increase exponentially with deadlines. Therefore, either the ratio ϕ for the packet scheduling problem can be achieved, or, to improve the lower bound, one would have to use non-monotone instances in the proof. Our competitive analysis uses a novel, intricate invariant technique involving dominance relations between sets of numbers, and is likely to find applications in the analysis of packet scheduling. We also show that the upper bound of ϕ for 3-bounded packet scheduling [6] extends to dynamic queues where all items have life span at most 3.

Finally, we address the case of dynamic queues and randomized algorithms. The algorithm RMix [6] can be easily adapted to dynamic queues, achieving competitive ratio of $e/(e-1)$ against an adaptive adversary. (In spite of the same value, this result is not related to our analysis of the uniform dynamic sets.) We prove a matching lower bound, which says that no memoryless randomized algorithm for dynamic queues can have competitive ratio smaller than $e/(e-1)$.

2 Preliminaries

We refer to the items currently in $\mathcal{S}$ as *active*. In other words, those are the items that have been already inserted but not yet deleted. Given some online

algorithm $\mathcal{A}$, an item is called *pending* for $\mathcal{A}$ if it is active but not yet collected by $\mathcal{A}$. We denote the weight of item x as w_x and the total weight of a set of items X as $w(X)$.

An instance of the problem is defined by a sequence I of item insertions or deletions. A solution consists of a *selection sequence* that specifies which item is selected at each step. An optimal solution can be computed in polynomial-time by reduction to maximum-weight matching: Represent the instance as a bipartite graph whose partitions are items and time steps. Each item a is connected to the time steps when this item is active with edges of weight w_a. Then the maximum-weight matching in this graph represents an optimal collection sequence.

Recall that when $\mathcal{S}$ is a dynamic queue, then $\mathcal{S}$ is represented by a list. We use symbol "$\lhd$" to represent the list ordering, that is $a \lhd b$ means that a is before b in the list. In case of dynamic queues, optimal solutions have special structure that we explore in our competitiveness proofs, namely they satisfy (without loss of generality) the following property:

> *The Earliest-Expiration-First (EEF) Property:* For any instance of the dynamic queue problem there is an optimal collection sequence with the following property: If a, b are active at the same time, $a \lhd b$, and both a, b are collected, then a is collected before b.

This property is easy to prove by a simple exchange argument: if b is selected at time t and a at time $t' > t$, then b is also active at time t', and therefore we can modify the collection sequence by first collecting a at time t and then b at time t'.

We use the standard definition of competitiveness. An online algorithm $\mathcal{A}$ is called $\mathcal{R}$-*competitive*, if for any instance I, the gain of $\mathcal{A}$ on I is at least the optimum gain on I divided by $\mathcal{R}$. (In the literature, an additive constant is sometimes also allowed in this bound. Since in our problem, all weights can be scaled up, this constant can be neglected; in all our bounds it is equal to 0.) The *competitive ratio* of $\mathcal{A}$ is the smallest $\mathcal{R}$ for which $\mathcal{A}$ is $\mathcal{R}$-competitive.

3 Dynamic Sets — Deterministic Algorithms

For general dynamic sets, it is quite easy to show the lower and upper bounds of 2 on the competitive ratio of deterministic online algorithms. We start with the lower bound.

Fact 1 *For dynamic sets, no deterministic online algorithm can be better than* 2*-competitive.*

Proof. We start with two items, say a and b with $w_a = w_b = 1$. By symmetry, we can assume that the algorithm first collects a. Then the adversary can collect b in the first step, delete it, and collect a in the next step, while the algorithm has no items to collect. □

We now show that this bound can be achieved.

Algorithm Greedy**.** At each step, if there is at least one pending item, collect the maximum-value pending item.

Fact 2 Greedy *is* 2*-competitive for dynamic sets.*

Proof. The proof is a straightforward adaptation of the proof for packet scheduling [11, 12] and we include it here only for the sake of completeness. We consider the items collected by Greedy and by an adversary. The items collected by the adversary are charged to the items collected by Greedy. Suppose the adversary collects an item b at time t. If this item was collected at time t or earlier by Greedy, we charge it to that item in Greedy's sequence. Otherwise, we charge it to the item collected by Greedy at time t. (Note that this item exists, since b is pending for Greedy at time t.)

Let a be an item collected by Greedy at some time u. It receives at most two charges: one from itself, if it was collected by the adversary at time u or later, and the other one from the item b collected by the adversary at time u. If a receives a charge from b, then b is pending for Greedy at time t, and therefore $w_b \leq w_a$. Therefore the charge to a is at most $2w_a$.

Summarizing, all adversary items are charged to our items, and each our item receives a charge of at most twice its weight. This implies that Greedy is 2-competitive. □

4 Dynamic Sets — Randomized Algorithms

In this section we study randomized algorithms for dynamic sets. In the randomized case, we can consider two types of the adversary: oblivious and adaptive.

For the adaptive adversary it is not hard to show a lower bound of 2. The adversary strategy is this: issue n items of weight 1. Collect any item a that is collected by the algorithm with probability at most $1/n$. Let b be the item collected by the algorithm. If $b = a$, remove all items. If $b \neq a$, remove all items except b and collect b. With probability at most $1/n$, we have $b = a$ and both the algorithm and the adversary collect b. With probability at least $1 - 1/n$, the algorithm gets one item and the adversary gets two items. So the ratio is arbitrarily close to 2.

In the sub-section below we show a lower bound of $\mathrm{e}/(\mathrm{e}-1)$ for randomized algorithms against the oblivious adversary. This lower bound applies even for uniform instances and decremental sets. Later we show that for decremental sets and uniform weights this bound is tight.

4.1 A Lower Bound for the Decremental Uniform Case

In this section we show that no randomized algorithm can achieve ratio better than $\mathrm{e}/(\mathrm{e}-1)$ for decremental sets, even in the uniform case – if all items have the same weight.

Fix any set size n and let $\mathcal{R} = \frac{n}{n+1} \cdot \frac{\mathrm{e}}{\mathrm{e}-1}$. The proof is based on Yao's min-max principle [18], i.e. we construct a probability distribution π over inputs, such that (i) the gain of an optimal solution is n on any input sequence from the support of π, and (ii) the expected gain of any deterministic algorithm run on a randomly chosen input from π is at most $n/\mathcal{R}$.

We construct the distribution π implicitly by the following random process: at each step, choose uniformly at random any active item a, collect it, and delete it after the step. Obviously, property (i) holds, and it is sufficient now to show (ii).

Fix any deterministic algorithm $\mathcal{A}$. Without loss of generality, $\mathcal{A}$ always collects an item if one is pending. Let $E_{a,p}$ be the expected number of items collected by $\mathcal{A}$ if a items are active and, among them, p items are pending. If $p = 0, 1$ then $\mathcal{A}$ collects p items. On the other hand, if $p > 1$, $\mathcal{A}$ collects an item in the first step, reducing the number of pending items to $p-1$, and then, with probability $(p-1)/a$ another pending element may get deleted. Hence, $E_{a,p}$ satisfies the following recurrence for any $a \geq 1$: $E_{a,0} = 0$, $E_{a,1} = 1$, and

$$E_{a,p} = \frac{a-p+1}{a} \cdot E_{a-1,p-1} + \frac{p-1}{a} \cdot E_{a-1,p-2} + 1$$

for $a \geq p \geq 2$.

The following technical lemma, can be used to bound the performance of any deterministic algorithm on π.

Lemma 1. *For all $a \geq p \geq 0$, we have $E_{a,p} \leq (a+1)(1-\mathrm{e}^{-p/a}) + 1$.*

Proof. We prove the lemma by induction on p. The case $p = 0$ holds trivially. For $a \geq p = 1$, $E_{a,p} = 1 \leq (a+1)(1-\mathrm{e}^{-1/a}) + 1$, as $(a+1)(1-\mathrm{e}^{-1/a}) > 0$. For $a \geq p \geq 2$, we use the inductive assumption and the recursive definition of $E_{a,p}$, getting

$$\begin{aligned}
E_{a,p} &= \frac{a-p+1}{a} \cdot E_{a-1,p-1} + \frac{p-1}{a} \cdot E_{a-1,p-2} + 1 \\
&\leq (a-p+1)\left(1-\mathrm{e}^{-\frac{p-1}{a-1}}\right) + (p-1)a\left(1-\mathrm{e}^{-\frac{p-2}{a-1}}\right) + 2 \\
&= a + 2 - \mathrm{e}^{-p/a} \cdot \left[(a-p+1)\mathrm{e}^{\frac{a-p}{a(a-1)}} + (p-1)\mathrm{e}^{\frac{2a-p}{a(a-1)}}\right] \\
&\leq a + 2 - \mathrm{e}^{-p/a} \cdot \left[(a-p+1)\left(1+\frac{a-p}{a(a-1)}\right) + (p-1)\left(1+\frac{2a-p}{a(a-1)}\right)\right] \\
&= (a+1)(1-\mathrm{e}^{-p/a}) + 1,
\end{aligned}$$

where the second inequality follows from $\mathrm{e}^x \geq x+1$. This completes the inductive step and the proof of the lemma. □

By Lemma 1, we have that $E_{n,n}$, the expected number of items collected by $\mathcal{A}$, is at most $(n+1)(\mathrm{e}-1)/\mathrm{e} + 1 = n/\mathcal{R} + O(1)$. Thus, property (ii) holds and, applying Yao's principle and taking the limit on n, we obtain our lower bound:

Theorem 1. *The competitive ratio of any randomized algorithm for the decremental uniform case of the item collection problem is at least* $\mathrm{e}/(\mathrm{e}-1)$.

4.2 An Upper Bound for the Decremental Uniform Case

We present a simple randomized algorithm with competitive ratio $\mathrm{e}/(\mathrm{e}-1)$, thus matching the lower bound from the previous section.

Algorithm UniRand. At each step collect with equal probability one of the pending items.

The idea behind the competitive analysis is to prove first that the optimal strategy of the adversary can be easily described: without loss of generality, at each step the adversary collects one item and deletes the same item. Then we analyze the competitive ratio of UniRand against such an adversary.

The number of items in the initial set is denoted by n. By a we denote the number of active items at a given step, and by $p \leq a$ the number of pending items. Note that p is a random variable. We consider states of the game conditioned on p being fixed, and we refer to such a state as a *configuration* (a, p). The definition of UniRand implies that in a configuration (a, p) each active item is pending with probability p/a.

We analyze the relation between the gain of UniRand and that of the adversary starting from each fixed configuration (a, p). Since the items are identical and the algorithm's pending items are distributed uniformly, we need not specify which items are deleted by the adversary in a given step, only their quantity.

We can split these steps into smaller parts—an *elementary* step consists in either collecting an item or deleting an item. We can thus describe the adversary's strategy as a sequence $S \in \{\mathtt{d}, \mathtt{c}\}^*$, where $\mathtt{c}$ means that the adversary takes an item (and allows the algorithm to take one as well) and $\mathtt{d}$ means that the adversary deletes an item. Not all such sequences are feasible. In the following we consider only the *feasible* strategies S for configuration (a, p), that is those in which: (i) the number of $\mathtt{d}$ operations in S is a, (ii) in every suffix of S the number of $\mathtt{c}$ operations is not greater than $\mathtt{d}$ operations.

By $E_{a,p}[S]$ we denote the expected gain of Algorithm UniRand starting from configuration (a, p), versus adversary strategy S.

Lemma 2. *For every a, p, S, it holds that $E_{a,p+1}[S] \geq E_{a,p}[S]$.*

Proof. We prove the inequality by induction on pairs (p, S), where $(p_1, S_1) < (p_2, S_2)$ if $|S_1| \leq |S_2|$ $|S_1| = |S_2|$ and $p_1 < p_2$. The inductive basis is straightforward: if $S = \mathtt{d}^a$ then $E_{a,p+1}[\mathtt{d}^a] = 0 = E_{a,p}[\mathtt{d}^a]$. If $p = 0$ then clearly $E_{a,1}[S] \geq 0 = E_{a,0}[S]$.

In the inductive step, we have two cases depending on the first action of S. If $S = \mathtt{c}S'$, then $E_{a,p+1}[\mathtt{c}S'] = 1 + E_{a,p}[S'] \geq 1 + E_{a,p-1}[S'] = E_{a,p}[\mathtt{c}S']$, by the inductive assumption. If $S = \mathtt{d}S'$, since we have $p + 1$ pending items, the adversary deletes a pending item with probability $\frac{p+1}{a}$ and a non-pending item with probability $\frac{a-p-1}{a}$. Using the inductive assumption, we get

$$\begin{aligned} E_{a,p+1}[\mathtt{d}S'] &= \frac{a-p-1}{a} E_{a-1,p+1}[S'] + \frac{p+1}{a} E_{a-1,p}[S'] \\ &\geq \frac{a-p-1}{a} E_{a-1,p}[S'] + \left[\frac{1}{a} E_{a-1,p}[S'] + \frac{p}{a} E_{a-1,p-1}[S']\right] \;=\; E_{a,p}[\mathtt{d}S'], \end{aligned}$$

□

Lemma 3. *Let $S = S_1\mathtt{cd}S_2$ be a feasible strategy for (a,p). If $S' = S_1\mathtt{dc}S_2$ is also a feasible strategy, then $E_{a,p}[S] \geq E_{a,p}[S']$.*

Proof. Suppose first that $S = \mathtt{cd}S_2$. If $p = 0$ then the claim is obvious, as $E_{a,0}[S] = E_{a,0}[S'] = 0$, and if $p = 1$ then $E_{a,1}[S] = 1 \geq E_{a,1}[S']$. So consider $p > 1$. By direct calculation:

$$\begin{aligned}
E_{a,p}[\mathtt{cd}S_2] &= 1 + E_{a,p-1}[\mathtt{d}S_2] \\
&= 1 + \frac{a-p+1}{a}E_{a-1,p-1}[S_2] + \frac{p-1}{a}E_{a-1,p-2}[S_2] \\
&= 1 + \frac{a-p}{a}E_{a-1,p-1}[S_2] + \frac{1}{a}E_{a-1,p-1}[S_2] + \frac{p-1}{a}E_{a-1,p-2}[S_2] \\
&\geq 1 + \frac{a-p}{a}E_{a-1,p-1}[S_2] + \frac{1}{a}E_{a-1,p-2}[S_2] + \frac{p-1}{a}E_{a-1,p-2}[S_2] \\
&= 1 + \frac{a-p}{a}E_{a-1,p-1}[S_2] + \frac{p}{a}E_{a-1,p-2}[S_2] \\
&= \frac{a-p}{a}E_{a-1,p}[\mathtt{c}S_2] + \frac{p}{a}E_{a-1,p-1}[\mathtt{c}S_2] \\
&= E_{a,p}[\mathtt{dc}S_2] \ .
\end{aligned}$$

Now consider the general case, when $S = S_1\mathtt{cd}S_2$ and $S' = S_1\mathtt{dc}S_2$. Then

$$\begin{aligned}
E_{a,p}[S] &= E_{a,p}[S_1\mathtt{cd}S_2] = \sum_i c_i E_{a-k,p-i}[\mathtt{cd}S_2] \\
E_{a,p}[S'] &= E_{a,p}[S_1\mathtt{dc}S_2] = \sum_i c_i E_{a-k,p-i}[\mathtt{dc}S_2]
\end{aligned}$$

for some k, $\{c_i\}$. By the previous calculations for all i, $E_{a-k,p-i}[\mathtt{cd}S_2] \geq E_{a-k,p-i}[\mathtt{dc}S_2]$ and hence

$$\begin{aligned}
E_{a,p}[S] = E_{a,p}[S_1\mathtt{cd}S_2] &= \sum_i c_i E_{a-k,p-i}[\mathtt{cd}S_2] \\
&\geq \sum_i c_i E_{a-k,p-i}[\mathtt{dc}S_2] = E_{a,p}[S_1\mathtt{dc}S_2] = E_{a,p}[S'],
\end{aligned}$$

completing the proof. □

We say that a strategy S for (a,a) is a *k-strategy* if it collects k items. Let $N_a = (\mathtt{cd})^a$ define the *natural* strategy for configuration (a,a), that is the one in which in each step we take an item and then delete it. The following lemma shows that any k-strategy for (a,a) is, in a sense, dominated by N_k on (k,k).

Lemma 4. *Let S be a k-strategy for (a,a). Then $E_{a,a}[S] \geq E_{k,k}[N_k]$.*

Proof. We apply Lemma 3. We know that in S there are k operations $\mathtt{c}$. Let $p_1 > p_2 > \ldots > p_k$ be their positions (counting from the right). As S is feasible

then no suffix contains more c operations than d operations, hence $p_i \geq 2i$. We iteratively modify S so that in the end in S one has $p_i = 2i$. Consider p_i for $i = 1, \ldots, k$. Suppose $p_i > 2i$. As $p_{i-1} = 2i - 2$ there are only d operations on positions $2i, 2i+1, \ldots, p_i - 1$. Then by Lemma 3 we can move c operation from position p_i to position $2i$ by consecutively inverting it with d operations. The value of $E_{a,a}[S]$ cannot increase by Lemma 3.

After ending the process we obtain another k-strategy $S' = \mathtt{d}^{a-k}(\mathtt{cd})^k$ and $E_{a,a}[S] \geq E_{a,a}[S']$. Clearly $E_{a,a}[\mathtt{d}^{a-k}(\mathtt{cd})^k] = E_{k,k}[(\mathtt{cd})^k] = E_{k,k}[N_k]$. □

We prove that the competitive ratio of the Algorithm UniRand is at most $\mathrm{e}/(\mathrm{e}-1)$.

Lemma 5. *For any integers $p \leq a$, it holds that $E_{a,p}[(\mathtt{cd})^a] \geq a(1-(1-1/a)^p)$.*

Proof. For $p = 0, 1$, $E_{a,p}[(\mathtt{cd})^a] = p$ and the lemma trivially holds. Also if $a = p = 2$, then $E_{a,p}[(\mathtt{cd})^a] = 3/2 = a(1-(1-1/a)^p)$.

For the remaining values of a and p we prove the lemma by induction on a. We note that for $p \geq 2$

$$E_{a,p}[(\mathtt{cd})^a] \;=\; \frac{a-p+1}{a} E_{a-1,p-1}[(\mathtt{cd})^{a-1}] + \frac{p-1}{a} E_{a-1,p-2}[(\mathtt{cd})^{a-1}] + 1 \;\;,$$

as the pending items are distributed uniformly among active items. Note that this is the same recurrence as in the proof of the lower bound. Thus, using the induction assumption, we get

$$\begin{aligned}
E_{a,p}[(\mathtt{cd})^a] &\geq \frac{a-p+1}{a}\,(a-1)\left(1-\left(\frac{a-2}{a-1}\right)^{p-1}\right) + \frac{p-1}{a}\,(a-1)\left(1-\left(\frac{a-2}{a-1}\right)^{p-2}\right) + 1 \\
&= a - \frac{(a-1)^2}{a}\left(\frac{a-2}{a-1}\right)^{p-2}\left(1+\frac{p-2}{(a-1)^2}\right) \\
&\geq a - \frac{(a-1)^2}{a}\left(\frac{a-2}{a-1}\right)^{p-2}\left(1+\frac{1}{(a-1)^2}\right)^{p-2} \\
&\geq a - \frac{(a-1)^2}{a}\left(\frac{a-1}{a}\right)^{p-2} \\
&= a\left(1-\left(\frac{a-1}{a}\right)^{p}\right),
\end{aligned}$$

completing the proof. □

Theorem 2. *Algorithm* UniRand *is* $\mathrm{e}/(\mathrm{e}-1)$*-competitive.*

Proof. Let k be the number of items collected by the adversary from n elements. Consider any k-strategy S for (n, n). By Lemmas 4,5, $E_{n,n}[S] \geq E_{k,k}[(\mathtt{cd})^k] \geq k(1-(1-1/k)^k)$. Therefore, the competitive ratio is at most

$$\frac{k}{E_{n,n}[S]} \;\leq\; \left[1-\left(1-\frac{1}{k}\right)^k\right]^{-1} \;<\; \frac{\mathrm{e}}{\mathrm{e}-1} \;\;,$$

□

5 Dynamic Queue — Deterministic Algorithms

In this section we consider deterministic online algorithms for the case when $\mathcal{S}$ is a dynamic queue. Thus now $\mathcal{S}$ is an ordered list and only items at the beginning of $\mathcal{S}$ (that is, a prefix) can be deleted. In the fully dynamic case, items can be inserted anywhere in the list, while in the decremental case, all items are present in the initial list and only prefix deletions are allowed.

Fact 3 *Every deterministic algorithm for queues has competitive ratio at least* $\phi \approx 1.618$.

Proof. We start with two items in the list, $a \lhd b$, with values $w_a = 1$ and $w_b = \phi$. If the algorithm chooses b, the adversary chooses a and deletes it, and in the next step he chooses b, so the ratio is $(w_a + w_b)/w_b = (1 + \phi)/\phi = \phi$. If the algorithm chooses a in the first step, the adversary chooses b and deletes both items, so the ratio is $w_b/w_a = \phi$ again. □

Note that this proof works for the decremental case. In fact, we only use two items with "life span" at most 2.

As explained in the introduction, the queue case generalizes the bounded-delay packet scheduling problem, or, equivalently, unit-job scheduling to maximize weighted throughput. The best known lower and upper bounds for this problem are ϕ and ≈ 1.83 [1, 7, 17, 8], respectively, and to determine the optimal bound remains a long-standing open problem.

This section includes some lower and upper bounds for the competitive ratio of deterministic online algorithms. Starting with lower bounds, we first prove that no deterministic algorithm for the dynamic queue – in fact, even for the decremental case – can achieve a competitive ratio better than 1.63. For memoryless algorithms we give a lower bound of 2.

Our first upper bound concerns decremental queues, for which we present a deterministic online algorithm DecQueEFH with competitive ratio ≈ 1.737. Next, we present a deterministic online algorithm FIFOQueEH that achieves competitive ratio 1.8 for FIFO Queues. Then we present the algorithm MarkAndPick that achieves competitive ratio ϕ for dynamic queues in which the item weights are non-decreasing (this also gives a ϕ-competitive algorithm for scheduling packets with non-decreasing weights). At the end of this section we show that the ϕ-competitiveness proofs for 3-bounded packet scheduling extend to dynamic queues.

5.1 Lower Bound of 1.63 for Decremental Queues

In this section we show that no online deterministic algorithm can have a competitive ratio smaller than 1.63 for decremental queues. The proof is by presenting an adversary's strategy that forces any deterministic online algorithm $\mathcal{A}$ to gain less than 1/1.63 times the adversary's gain.

Adversary's strategy. To get a cleaner analysis, we first present the argument for the dynamic case (with insertions allowed), and explain later how it extends to the decremental case. We assume that the items appear gradually, so that at each step the algorithm has at most three items to choose from.

Fix some $n \geq 2$. To simplify notation, in this section we refer to items simply by their weight, thus below "z_i" denotes both an item and its weight. The instance consists of a sequence of $2n$ items $1, z_1, z_2, ..., z_{2n-2}, z_{2n}$ (note that the item indexed $2n-1$ is not included) such that

$$z_2 \lhd z_4 \lhd ... \lhd z_{2n-2} \lhd z_{2n} \lhd z_{2n-3} \lhd ... \lhd z_3 \lhd z_1 \lhd 1, \quad \text{and}$$

$$1 > z_1 > z_2 > \ldots > z_{2n-3} > z_{2n-2} > z_{2n} > 0.$$

The even- and odd-numbered items in this sequence form two roughly geometric sequences. In fact, z_{2i} is only slightly smaller than z_{2i-1}, for all $i = 1, ..., n-1$.

Initially, items $z_2 \lhd z_1 \lhd 1$ are present. Generally, in step $i = 1, 2, ..., n-1$, the adversary maintains the invariant that the active items are $z_{2i} \lhd z_{2i-1} \lhd ... \lhd 1$, of which only three items z_{2i}, z_{2i-1} and 1 are pending for $\mathcal{A}$ (that is, $\mathcal{A}$ already collected $z_{2i-3}, ..., z_1$). The adversary's move depends now on what $\mathcal{A}$ collects in this step:

(i) $\mathcal{A}$ collects z_{2i}. Then the adversary ends the game by deleting all active items. In this case the adversary collects i heaviest items: $1, z_1, z_2, z_3, \ldots, z_{i-1}$.

(ii) $\mathcal{A}$ collects 1. The adversary ends the game by deleting x_{2i} and x_{2i-1}. This leaves $\mathcal{A}$ with no pending items, and the adversary can now collect $\mathcal{A}$'s items one by one. Overall, in this case the adversary collects $2i$ heaviest items: $1, z_1, z_2, \ldots, z_{2i-2}, z_{2i-1}$.

(iii) $\mathcal{A}$ collects z_{2i-1}. In this case the game continues. If $i < n-1$, the adversary deletes z_{2i}, inserts z_{2i+2} and z_{2i+1} into the current list (according to the order defined earlier), and we go to step $i+1$. The case $i = n-1$ is slightly different: here the adversary only inserts the last item z_{2n} before proceeding to step n (described below).

If the game reaches step n, $\mathcal{A}$ has two pending items, z_{2n} and 1. In this step, the adversary behavior is similar to previous steps: if $\mathcal{A}$ collects z_{2n}, then the adversary deletes the whole sequence and collects n heaviest items: $1, z_1, z_2, z_3, \ldots, z_{n-1}$. If $\mathcal{A}$ collects 1, the adversary deletes z_n, leaving $\mathcal{A}$ without pending items, and allowing the adversary to collect the whole sequence.

Our goal is to find a sequence $\{z_i\}$, as described above, and a constant $\mathcal{R}$ such that

$$\mathcal{R} \cdot (1 + \textstyle\sum_{i=1}^{j} z_{2i-1}) \leq 1 + \sum_{i=1}^{2j+1} z_i \qquad \text{for all } 0 \leq j < n \tag{1}$$

$$\mathcal{R} \cdot (z_{2j+2} + \textstyle\sum_{i=1}^{j} z_{2i-1}) \leq 1 + \sum_{i=1}^{j} z_i \qquad \text{for all } 0 \leq j < n \tag{2}$$

Lemma 6. *Suppose that there is a sequence $1, z_1, ..., z_{2n-2}, z_{2n}$, and a constant $\mathcal{R}$ that satisfy inequalities (1) and (2). Then there is no $\mathcal{R}$-competitive deterministic online algorithm for dynamic queues, even in the decremental case.*

The lemma should be clear from the description of the strategy given earlier, since the sums in inequalities (1) and (2) represent the gains of the adversary and the algorithm in various steps. The only part that needs justification is that the lemma holds in the decremental case. To see this, we slightly modify the adversary strategy: The sequence $\{z_i\}$ is created all at once in the beginning, and whenever $\mathcal{A}$ deviates from the choices (i), (ii), (iii), it must be collecting an item lighter than z_{2i}, and thus the adversary can finish the game as in Case (i).

Analysis for 6 *items.* We now exhibit a sequence of 6 items for which Lemma 6 holds with $\mathcal{R} \approx 1.63$. To simplify notation, we rename the items: z_2, z_4, z_6, z_3, z_1 as x, y, z, u, v, respectively. Otherwise we follow the aforementioned idea.

By Lemma 6, we want to find numbers x, y, z, u, v such that $0 < z < y < u < x < v < 1$ and a maximal $\mathcal{R}$ for which:

$$\begin{aligned} \mathcal{R} \cdot x &\leq 1 \\ \mathcal{R} \cdot 1 &\leq 1 + v \\ \mathcal{R} \cdot (v + y) &\leq 1 + v \\ \mathcal{R} \cdot (1 + v) &\leq 1 + v + x + u \\ \mathcal{R} \cdot (v + u + z) &\leq 1 + v + x \\ \mathcal{R} \cdot (1 + v + u) &\leq 1 + v + x + u + z \end{aligned}$$

We can solve it by replacing inequalities by equations, and after doing substitutions, the problem reduces to finding a solution of a polynomial equation $x^5 + x^4 + 5x^3 - x^2 - 1 = 0$. This polynomial has exactly one real root, $x = 0.61238...$, which yields $\mathcal{R} = 1.6329....$ Summarizing, we get the following result:

Theorem 3. *There is no deterministic online algorithm for dynamic queues (even in the decremental case) with competitive ratio smaller than* 1.6329*.*

A natural question is how much this ratio can be improved with sequences $\{z_i\}$ of arbitrary length. For example for $n = 5$ (10 items) one can obtain $\mathcal{R} = 1.6367...$ and our experiments indicate that the corresponding ratios tend to ≈ 1.6378458, so the improvement is minor.

Interestingly, almost the same ratio 1.63784 appeared in [9] for what appears to be an unrelated problem of scheduling unit jobs (without weights), with rejections, for minimizing the total completion time. In that paper, as in our case, this constant is also defined implicitly, although in a very different way. To those fascinated by mysterious connections between seemingly unrelated phenomena, we want to point out that an ever closer number 1.637845 appears (with a minus sign) on page S11 in [5].

5.2 Lower Bound of 2 for Memoryless Algorithms

A memoryless algorithm decides which item to collect based only on the weights of its pending items. We now give a lower bound for such memoryless algorithms.

Theorem 4. *For dynamic queues, no memoryless algorithm has competitive ratio smaller than* 2.

Proof. Fix a memoryless algorithm $\mathcal{A}$. We give an adversary's strategy where the adversary's gain is $2 - o(1)$ times $\mathcal{A}$'s gain.

Pick large integers n and $T \gg n$, and let $X = \{x_0, ..., x_n\}$ where $w_{x_i} = 1 + \frac{i}{n}$ for $i = 0, 1, ..., n$. The adversary maintains the invariant that at each step $\mathcal{A}$'s pending set is X, with the items ordered by increasing value. Suppose that for this pending set $\mathcal{A}$ collects some item x_k.

If $k = 0$, the adversary collects item x_n, deletes all items, inserts copies of all items from X again into the queue, and repeats the process T times. $\mathcal{A}$'s gain is $Tw_{x_0} = T$ while the optimum gain is $Tw_{x_n} = 2T$, so the ratio is 2.

Suppose now that $k \geq 1$. In this case, the adversary collects x_{k-1}, deletes all items $x_0, ..., x_{k-1}$ for $i = 0, 1, ..., k-1$ and inserts new copies of items $x_0, ..., x_k$. This process is repeated T times. After T steps, the adversary collects the remaining uncollected items, in particular, all T copies of item x_k. $\mathcal{A}$ can of course collect the remaining pending items. The value collected by $\mathcal{A}$ is at most $Tw_{x_k} + 2(n+1) = T(1 + k/n) + 2(n+1)$, while the value collected by the adversary is at least $T(w_{x_{k-1}} + w_{x_k}) = T(2 + (2k-1)/n)$. So with $T = n^3$ and $n \to \infty$ the ratio approaches 2. □

5.3 Upper Bound of 1.737 for Decremental Queues

We now give a ≈ 1.737-competitive deterministic algorithm for decremental queues.

Algorithm DecQueEFH*:* The computation is divided into stages, where each stage is a single step, a pair of consecutive steps, or a triple of consecutive steps. By h we denote the maximum-weight pending item from the first step of the stage. We use two parameters, $\beta = (\sqrt{13}+1)/8$ and $\xi = (\sqrt{13}+1)/6$. Note that $\beta < \xi$. Without loss of generality, we assume that there are always pending items, for we can always insert any number of 0-weight items into the instance, without changing the competitive ratio. In the pseudo-code below, we assume that after each item collection the algorithm proceeds to the next step of the process.

(E) let h be the heaviest pending item
collect the earliest pending item e with $w_e \geq \beta w_h$
(F) if h is not pending then end stage and goto (E)
collect the earliest item f with $w_f \geq \xi w_h$
(H) if h is not pending then end stage and goto (E)
collect h
end stage and goto (E)

Theorem 5. *For decremental queues, the competitive ratio of* DecQueEFH *is at most* $\mathcal{R} = 2(\sqrt{13} - 1)/3 \approx 1.737$.

Proof. We fix an instance and we compare DecQueEFH's gain on this instance to the adversary's gain. Without loss of generality, we assume that the adversary has the EEF property.

The proof is by amortized analysis. We preserve the invariant that, after each stage, each item i that is pending for the adversary but has already been collected by DecQueEFH has credit associated with it of value equal w_i. The adversary's amortized gain is equal to his actual gain plus the total credit change. To prove the theorem, it is then sufficient to prove the following claim:

$(*)$ In each stage the adversary's amortized gain is at most $\mathcal{R}$ times DecQueEFH's gain.

To prove $(*)$, we consider several cases depending on the number of steps in a stage and on the location of items collected by the adversary relatively to those collected by DecQueEFH. We assume that at each step the adversary collects items that are not collected by the algorithm before or during this stage. Otherwise, either the adversary collects an item that has been collected by DecQueEFH earlier and has credit on it that can pay for the adversary gain for this item, or DecQueEFH collected this item in this stage, in which case we can think of the algorithm giving the adversary credit for this item anyway (even though it is not needed, for this item is not pending for the adversary anymore).

We first observe that $e \trianglelefteq f \trianglelefteq h$ (if f is defined for this stage). This follows immediately from $\beta < \xi < 1$. An important consequence of this, that plays a major role in the argument below, is that when h is deleted, then e and f are deleted as well.

Case 1: h is not pending in (F). The stage has only one step, and the algorithm collects an item e with $w_e \geq \beta w_h$. Let a be the item collected by the adversary.

If h was deleted after (E) then e was deleted as well. Thus we do not give the adversary credit for e, and his amortized gain is $w_a \leq w_h$. The ratio is

$$\frac{w_a}{w_e} \leq \frac{1}{\beta} \;=\; \mathcal{R}.$$

Suppose that h was collected (that is, $e = h$). If $a \triangleleft h$ then $w_a \leq \beta w_h$, so, together with the credit for h, the amortized adversary's gain is $w_a + w_h \leq (\beta + 1)w_h$, and the ratio is

$$\frac{w_a + w_h}{w_h} \leq 1 + \beta \;<\; \mathcal{R}.$$

If $h \triangleleft a$, we need not give the adversary any credit, so the ratio is at most 1.

Case 2: h is pending in (F) but is not pending in (H). The stage has two steps, and we collect e and f, gaining $w_e + w_f \geq (\beta + \xi)w_h$. The adversary collects two items, say a and b with $a \triangleleft b$.

If h was deleted after (F), then both e and f are deleted as well, so we do not give the adversary any credits. Thus the adversary amortized gain is $w_a + w_b \leq 2w_h$. In this case the ratio is

$$\frac{w_a + w_b}{w_e + w_f} \leq \frac{2}{\beta + \xi} \;=\; 4(\sqrt{13} - 1)/7 \;<\; \mathcal{R}.$$

Suppose now that the algorithm collected h in (F) (that is, $f = h$), and thus our gain is actually $w_e + w_h \geq (1+\beta)w_h$. If $b \lhd e$, then the adversary amortized gain is $w_a + w_b + w_e + w_h \leq (2\beta+1)w_h + w_e$. Thus the ratio is

$$\frac{w_a + w_b + w_e + w_h}{w_e + w_h} \leq \frac{3\beta+1}{\beta+1} \;=\; 4(\sqrt{13}+10)/17 \;<\; \mathcal{R}.$$

If $e \lhd b \lhd h$, the adversary amortized gain is $w_a + w_b + w_h \leq (2\xi+1)w_h$, so the ratio is

$$\frac{w_a + w_b + w_h}{w_e + w_h} \leq \frac{2\xi+1}{\beta+1} \;=\; 2(5\sqrt{13}+23)/51 \;<\; \mathcal{R}.$$

If $h \lhd b$, we need not give the adversary any credit, gaining $w_a + w_b \leq 2w_h$. So the ratio is less than the one above because $2\xi + 1 > 2$.

Case 3: h is still pending in (H). In this case the stage has three steps and we collect e, f, and h, for the total gain of $w_e + w_f + w_h \geq (\beta + \xi + 1)w_h$. The adversary collects three items $a \lhd b \lhd c$ and may get credit for other.

If $c \lhd e$, then the adversary gets credit for e, f and h, and his amortized gain is $w_a + w_b + w_c + w_e + w_f + w_h \leq 3\beta w_h + w_e + w_f + w_h$. So the ratio is

$$\frac{w_a + w_b + w_c + w_e + w_f + w_h}{w_e + w_f + w_h} \leq \frac{4\beta + \xi + 1}{\beta + \xi + 1} \;=\; \mathcal{R}.$$

If $e \lhd c \lhd f$, then the adversary gets credit for f, h, and his amortized gain is $w_a + w_b + w_c + w_f + w_h \leq 3\xi w_h + w_f + w_h$. So the ratio is

$$\frac{w_a + w_b + w_c + w_f + w_h}{w_e + w_f + w_h} \leq \frac{4\xi + 1}{\beta + \xi + 1} \;=\; \mathcal{R}.$$

If $f \lhd c \lhd h$, then he only gets credit for h, so his amortized gain is $w_a + w_b + w_c + w_h \leq 4w_h$. The ratio is

$$\frac{w_a + w_b + w_c + w_h}{w_e + w_f + w_h} \leq \frac{4}{\beta + \xi + 1} \;=\; 8(31 - 7\sqrt{13})/27 \;<\; \mathcal{R}.$$

Finally, when $h \lhd c$, the adversary amortized gain is $w_a + w_b + w_c \leq 3w_h$, which is less than in the previous case. □

The analysis of DecQueEFH is tight. More precisely, we show that the competitive ratio of DecQueEFH is at least $2(\sqrt{13}-1)/3 \approx 1.737$, regardless of the choice of β and ξ. In fact its competitive ratio cannot be improved by adding further stages with non-decreasing parameters.

Consider three instances. In the first instance, we have $a \lhd b$ with weights $w_a = \beta$, $w_b = 1$. DecQueEFH collects a, while the adversary collects b and deletes the whole sequence. The ratio is $\mathcal{R}_1 = \frac{1}{\beta}$.

In the second instance, $a \lhd b \lhd c \lhd d \lhd e \lhd f$, with $w_a = w_b = w_c = \beta - \epsilon$, $w_d = \beta$, $w_e = \xi$, $w_f = 1$. In the first three steps DecQueEFH collects d, e and f, while the adversary collects a, b and c. Right after the first step the adversary

deletes a, b and c, and in the remaining steps collects d, e and f. The ratio is arbitrarily close to $\mathcal{R}_2 = \frac{4\beta+\xi+1}{\beta+\xi+1}$.

In the third instance, $a \lhd b \lhd c \lhd d \lhd e \lhd f$, with $w_a = \beta$, $w_b = w_c = w_d = \xi - \epsilon$, $w_e = \xi$, $w_f = 1$. In the first three steps DecQueEFH collects a, e and f, while the adversary collects b, c and d. Right after the third step the adversary deletes b, c and d, and in the remaining steps collects e and f. The ratio is arbitrarily close to $\mathcal{R}_3 = \frac{4\xi+1}{\beta+\xi+1}$.

If $\beta > \frac{3}{4}$, it holds that

$$\mathcal{R}_2 = \frac{4\beta+\xi+1}{\beta+\xi+1} = 1 + \frac{3\beta}{\beta+\xi+1} > 1 + \frac{\frac{9}{4}}{\frac{7}{4}+\xi} \geq 1 + \frac{\frac{9}{4}}{\frac{7}{4}+1} = 1 + \frac{9}{11} = 1.818...$$

On the other hand, if $\beta \leq \frac{3}{4}$, by setting $\xi = \frac{4}{3}\beta$ we obtain

$$\mathcal{R}_2 = \mathcal{R}_3 = \frac{16\beta+3}{7\beta+3} \ ,$$

minimizing the maximum of $\mathcal{R}_2$ and $\mathcal{R}_3$, as the former is a decreasing and the latter an increasing function of ξ, while $\mathcal{R}_1$ does not depend on ξ. Likewise, with $\xi = \frac{4}{3}\beta$, $\mathcal{R}_1$ is a decreasing and $\mathcal{R}_2 = \mathcal{R}_3$ an increasing function of β, so their maximum is minimized when $\mathcal{R}_1 = \mathcal{R}_2 = \mathcal{R}_3$ holds. This gives the equation $16\beta^2 - 4\beta - 3 = 0$, with a sole positive solution $\beta = (\sqrt{13}+1)/8$ and $\xi = \frac{4}{3}\beta = (\sqrt{13}+1)/6$. These are the values we used in the algorithm, and they yield the competitive ratio of $2(\sqrt{13}-1)/3 \approx 1.737$.

Note that adding further stages with parameters at least as large ξ would not change DecQueEFH's choices in any step, so the choice of three stages is optimal as well.

5.4 Upper Bound of 1.8 for FIFO Queues

We now generalize the idea of the previous algorithm so that it works for FIFO queues. The algorithm uses two parameters, α and β. The main idea is this: If the heaviest item h' from the previous step is no longer pending, this is fine, as it simply begins another step. If, however, h' is still pending and new items have been inserted to the queue, the algorithm inspects them. If the heaviest new item h is not too heavy (that is, if $\alpha w_h \leq w_{h'}$), the algorithm ignores new items and collects h'. If, however, h is very heavy ($\alpha w_h > w_{h'}$), the algorithm forgets about h', resets *heavy* to h and collects the earliest pending item e, such that $w_e \geq \beta w_h$.

Algorithm FIFOQueEH*:* By h we denote the maximum-weight pending item and by h' the previous maximum-weight pending item (initially h' is an imaginary item of weight 0). We use two parameters, $0 < \alpha, \beta < 1$. Without loss of generality, we assume that there are always pending items, for we can always insert any number of 0-weight items into the instance, without changing the competitive ratio. In the pseudo-code below, we assume that after each item collection the algorithm proceeds to the next step of the process.

let h be the heaviest pending item and h' the previous heaviest item
(E) if (h' is not pending) or (h' is pending and $\alpha w_h \geq w_{h'}$) then collect the earliest item e with $w_e \geq \beta w_h$
(H) else collect h'

We define *stages* of the algorithm. The first stage begins before FIFOQueEH collects any item. Each next stage begins immediately after previous stage ends. The stage ends when h' is deleted by the adversary or condition (H) holds. The last step of the stage is the last step t such that at beginning of t, h' was pending for FIFOQueEH.

Let $e_1, e_2, \ldots e_k$ be the set of items collected (in this order) by FIFOQueEH in one stage when (E) condition held. Let $h_1, \ldots, h_k$ be the corresponding heavy items. From the algorithm and the definition of FIFO queues, we have:

Fact 4 *For all $i = 1, ..., k$ we have $e_i \trianglelefteq h_i$ (with all relations strict, except possibly for $i = k$) and $h_i \trianglelefteq h_{i+1}$.*

Theorem 6. *The competitive ratio of* FIFOQueEH *with $\alpha = \frac{3}{4}$ and $\beta = \frac{2}{3}$ is at most* 1.8.

Proof. The proof is by amortized analysis. We preserve the invariant that, after each stage, each item i that is pending for the adversary but has already been collected by FIFOQueEH has credit associated with it of value equal w_i. The adversary's amortized gain is equal the his actual gain plus the total credit change. To prove the theorem, it is then sufficient to prove the following claim:

($*$) In each stage the adversary's amortized gain is at most 1.8 times FIFOQueEH's gain.

We prove ($*$) for each of the following cases (1) the stage ends because the heaviest item is deleted, or (2) the stage ends because FIFOQueEH collects h'. The second case has three sub-cases, depending on which condition the latest item a collected by the adversary satisfies: (2a) $a \triangleleft e_l$ for some $l \leq k$, (2b) $e_k \trianglelefteq a \triangleleft h_k$, or (2c) $h_k \trianglelefteq a$.

Case 1: The stage ended because item h was deleted. The adversary cannot gain credit for any items taken by the algorithm, as they are not pending after that stage. Hence the gain of the adversary is at most $\sum_{i=1}^{k} w_{h_i}$. The gain of the algorithm is $\sum_{i=1}^{k} w_{e_i}$. As $w_{e_i} \geq \beta w_{h_i}$ for all i, the competitive ratio is at most $\frac{1}{\beta} = 1.5$.

Case 2: The stage ends because the new heaviest item was light, that is $\alpha w_{h_{k+1}} < w_{h_k}$.

Case 2a: Suppose that $a \triangleleft e_l$ for some $l \leq k$, choose minimal such l. Let $c_i = w_{e_i} - \beta w_{h_i}$ for $i = 1, \ldots, l-1$. We estimate the amortized adversary's gain: in step $i \leq l-1$ the heaviest pending item is h_i, and in step $i \in \{l \ldots k+1\}$ the adversary collects item preceding e_l. If this item is still pending for the algorithm, its weight is at most βw_{h_l}. If it is not pending for the algorithm, it is one of the e_i's, for some $i < l$. Then the gain of the adversary is $w_{e_i} = \beta w_{h_i} + c_i < \beta w_{h_l} + c_i$. As we can collect each such e_i only once, all these steps add to at

most $(k-l+2)\beta w_{h_l} + \sum_{i=1}^{l-1} c_i$. The adversary gets credit for all items taken by the algorithm in steps $l, \ldots, k+1$ that is $\sum_{i=l}^{k} w_{e_i} + w_{h_k}$. Thus the amortized gain of the adversary is at most

$$\sum_{i=1}^{l-1} w_{h_i} + (k-l+2)\beta w_{h_l} + \sum_{i=1}^{l-1} c_i + \sum_{i=l}^{k} w_{e_i} + w_{h_k} \ ,$$

On the other hand the gain of the algorithm is

$$\sum_{i=1}^{k} w_{e_i} + w_{h_k} = \sum_{i=1}^{l-1} (\beta w_{h_i} + c_i) + \sum_{i=l}^{k} w_{e_i} + w_{h_k} \ .$$

Hence for fixed l, the competitive ratio of FIFOQueEH in the stage is at most

$$\mathcal{R}_{1,l} = \frac{\sum_{i=1}^{l-1} h_i + (k-l+2)\beta h_l + \sum_{i=1}^{l-1} c_i + \sum_{i=l}^{k} e_i + h_k}{\sum_{i=1}^{l-1} \beta h_i + \sum_{i=1}^{l-1} c_i + \sum_{i=l}^{k} e_i + h_k} \ .$$

Fact 5 *For any $\gamma, x, m > 0$ and $n > x$, either $\frac{n}{m} < \frac{1}{\gamma}$ or $\frac{n-x}{m-\gamma x} \geq \frac{n}{m}$.*

We upper-bound $\mathcal{R}_{1,l}$. All the following inequalities follow from Fact 5 (for $\gamma = 1$ or $\gamma = \beta$):

$$\begin{aligned}
\mathcal{R}_{1,l} &\leq \frac{\sum_{i=1}^{l-1} h_i + (k-l+2)\beta h_l + \sum_{i=l}^{k} e_i + h_k}{\sum_{i=1}^{l-1} \beta h_i + \sum_{i=l}^{k} e_i + h_k} \\
&\leq \frac{\sum_{i=1}^{l-1} h_i + (k-l+2)\beta h_l + \sum_{i=l}^{k} \beta h_i + h_k}{\sum_{i=1}^{l-1} \beta h_i + \sum_{i=l}^{k} \beta h_i + h_k} \\
&\leq \frac{(k-l+2)\beta h_l + \sum_{i=l}^{k} \beta h_i + h_k}{\sum_{i=l}^{k} \beta h_i + h_k} \\
&= 1 + \frac{(k-l+2)\beta h_l}{\sum_{i=l}^{k} \beta h_i + h_k} \ .
\end{aligned}$$

Fix k, l and h_l. The fraction above is maximal when $h_{l+1}, \ldots h_k$ are minimal. As $\alpha h_{i+1} \geq h_i$, minimal $h_l, \ldots h_k$ form a geometric progression with a common ratio of α^{-1}. Thus by taking $h_i = \alpha^{k-i} h_k$ for $i \geq l$ and fixing $m := k-l+1 \geq 1$, we obtain

$$\mathcal{R}_{1,l} \leq 1 + \frac{(k-l+2)\beta h_l}{\sum_{i=l}^{k} \beta h_i + h_k} \leq 1 + \frac{(m+1)\beta\alpha^{m-1}}{\sum_{i=0}^{m-1} \beta\alpha^i + 1} = \mathcal{B}_m \ .$$

For $m = 1$ we have $\mathcal{B}_1 = 1 + 2\beta/(1+\beta) = 1.8$, and for $m = 2$ we have $\mathcal{B}_2 = 1 + 3\beta\alpha/(\beta(1+\alpha)+1) = 1 + 9/13 < 1.8$. As for $m \geq 3$, we show that $\mathcal{B}_m < \mathcal{B}_{m-1}$:

$$\begin{aligned}
\mathcal{B}_m - 1 = \frac{(m+1)\beta\alpha^{m-1}}{\beta\sum_{i=0}^{m-1}\alpha^i + 1} &< \frac{(m+1)\beta\alpha^{m-1}}{\beta\sum_{i=0}^{m-2}\alpha^i + 1} = \\
&= \alpha \cdot \frac{m+1}{m} \cdot \frac{m\beta\alpha^{m-2}}{\beta\sum_{i=0}^{m-2}\alpha^i + 1} \leq \frac{m\beta\alpha^{m-2}}{\beta\sum_{i=0}^{m-2}\alpha^i + 1} = \mathcal{B}_{m-1} - 1 \ .
\end{aligned}$$

Thus all $\mathcal{R}_{1,l}$ are upper-bounded by 1.8.
Case 2b: Suppose $e_k \trianglelefteq a \lhd h_k$. The adversary collected only the items preceding h_k in the queue, thus gaining at most $h_1+\ldots+h_k+2h_k$, as in step $i \in \{1 \ldots k-1\}$ it can collect item of weight greater than h_i, in steps $k, k+1$ it can collect items with weight almost h_k and gain credit for h_k. Hence the competitive ratio of FIFOQueEH can be upper-bounded by

$$\mathcal{R}_2 = \frac{\sum_{i=1}^{k} h_i + 2h_k}{\sum_{i=1}^{k} e_i + h_k} \leq \frac{\sum_{i=1}^{k} h_i + 2h_k}{\sum_{i=1}^{k} \beta h_i + h_k} = \mathcal{B}$$

and by Fact 5 for $\gamma = \beta$ either $\mathcal{B} < \frac{1}{\beta} = 1.5$ or

$$\mathcal{B} \leq \frac{h_k + 2h_k}{\beta h_k + h_k} = \frac{3}{1+\beta} = 1.8\ .$$

Case 2c: Suppose $h_k \trianglelefteq a$. Then he cannot get credit for any item taken by the algorithm, due to the EEF property. In this case the adversary's gain is at most $h_1 + \ldots + h_{k+1}$, which gives smaller competitive ratio than the competitive ratio obtained in case (2b), as $\alpha > \frac{1}{2}$. This concludes the proof. □

The analysis of FIFOQueEH is tight. More precisely, we show that the competitive ratio of Algorithm FIFOQueEH is at least 1.8, regardless of the choice of α and β.

We consider two instances. In the first one, we have $a \lhd b \lhd c \lhd d$, with $w_a = w_b = \beta - \epsilon$, $w_c = \beta$, $w_d = 1$. Item a is deleted right after the first step, and item b right after the second step. FIFOQueEH collects c in the first step and d in the second step, while the adversary collects all four items. Adversary's gain is $1 + 3\beta - 2\epsilon$ and the FIFOQueEH's gain is $1 + \beta$. Thus the competitive ratio is arbitrarily close to $(3\beta + 1)/(\beta + 1)$.

In the second instance, $a \lhd b \lhd c \lhd d$, with $w_a = \beta$, $w_b = w_c = 1 - \epsilon$, $w_d = 1$. Items a and b are deleted right after the first step, and c is deleted right after the second step. FIFOQueEH collects a in the first step and d in the second step, so its gain is $1 + \beta$. The adversary collects b, c, d (in this order), so his gain is $3 - 2\epsilon$. Thus the competitive ratio is arbitrarily close to $3/(1 + \beta)$.

From these two instances, we get that the ratio of FIFOQueEH is at least $\max\{3\beta + 1, 3\}/(\beta + 1)$, and this quantity is at least 1.8 for any β.

5.5 Upper Bound ϕ for Non-Decreasing Weights

In this section we give an online algorithm MarkAndPick that is ϕ-competitive for dynamic queues when item weights are increasing. More precisely, if $\lhd$ denotes the ordering of the items in the queue, then we assume that, at any time, for any two active items $a, b \in \mathcal{S}$, if $a \lhd b$ then $w_a \leq w_b$. (Recall that the *active* items are those currently in $\mathcal{S}$ and the *pending* items are those items in $\mathcal{S}$ that have not been yet collected by the algorithm.)

Algorithm MarkAndPick*:* At each step $t = 1, 2, ...$, if there are no pending items, wait. Otherwise, let h be the heaviest unmarked item (not necessarily active). Mark h and collect the earliest (lightest) pending item i with $w_i \geq w_h/\phi$.

We need not mark deleted items (those not in $\mathcal{S}$), for whenever $h \notin \mathcal{S}$ then, by the weight ordering, the item we collect is at least as heavy as h, and thus we simply collect the earliest pending item. We state the algorithm in the above form to simplify the analysis.

Set dominance relation. Let X, Y be two finite sets of numbers. We say that X *dominates* Y, denoted $X \succeq Y$, if either $Y = \emptyset$ or $\max X \geq \max Y$ and $(X - \max X) \succeq (Y - \max Y)$. Note that we do not require that $|X| = |Y|$. In particular, $X \succeq \emptyset$.

For any set T and a number u, let $\sharp_u(T) = |\{t \in T : t \geq u\}|$. We show that the majorization can be described in terms of $\sharp_u$ The following lemma is routine and we omit the proof.

Lemma 7. *The following three conditions are equivalent:*

(i) $X \succeq Y$,
(ii) There is an injection $f : Y \to X$ *such that* $f(y) \geq y$ *for all* y.
(iii) For every x *we have* $\sharp_x(X) \geq \sharp_x(Y)$.

Lemma 8. *Suppose that* $X \succeq Y \neq \emptyset$. *Then*

(i) $X - \min X \succeq Y - \min Y$.
(ii) If $x \in X \cap Y$ *then* $X - x \succeq Y - x$.
(iii) If $X, Y \subseteq Z$, $y \in Z - Y$, *and* $x \geq \max\{z \in Z - X : z \leq y\}$ *then* $X \cup x \succeq Y \cup y$. *(In particular, this holds for* $x \geq y$.*)*

Proof. Parts (i) and (ii) are straightforward, so we only show (iii). Let $x' = \max\{z \in Z - X : z \leq y\}$. It is sufficient to show (iii) for $x = x'$.

We use Lemma 7(iii). For $u \leq x'$, since $X \succeq Y$, we have $\sharp_u(X \cup x') = \sharp_u(X) + 1 \geq \sharp_u(Y) + 1 = \sharp_u(Y \cup y)$. For $u > y$, we have $\sharp_u(X \cup x') = \sharp_u(X) \geq \sharp_u(Y) = \sharp_u(Y \cup y)$. Suppose $x' < u \leq y$. Since $y \in Z - Y$ and $X \cap [u, y] = Z \cap [u, y]$, we have $|X \cap [u, y]| > |Y \cap [u, y]|$, and therefore $\sharp_u(X \cup x') = \sharp_u(X) \geq \sharp_u(Y) + 1 = \sharp_u(Y \cup y)$. □

For simplicity, we assume that all weights are different. If there are equal weights, we can perturb them slightly or extend the weight ordering using the item indices. For sets B and C of items, we say that B dominates C, writing $B \succeq C$, if $\{w_b : b \in B\}$ dominates $\{w_c : c \in C\}$. We write $B \succeq aC$ if $\{w_b : b \in B\}$ dominates $\{aw_c : c \in C\}$.

We now consider the behavior of the adversary. By the EEF property, for any $i, j \in \mathcal{S}$, we can assume that if j is the item collected in step t and $i \lhd j$ then the adversary will not collect item i in the future. Thus, instead of considering the whole set of pending adversary items, we can restrict ourselves only to those that are after the one he collected last. Let C_t be the set of these items. We update C_t as follows: at each step, the deleted items are removed from C_t and

released items are added to C_t. Then, if the adversary collects an item j, then we remove all items $i \lhd j$ from C_t.

Basic idea: The proof is based on a charging scheme, where the items collected by the adversary are charged to our items in such a way that each our item is charged at most ϕ times its weight. In other words, each our item i has a budget ϕw_i and it uses its budget to pay for some items in the adversary's set. In fact, each such i pays for either one or two adversary's items. The charging is done in two steps:

(1) we charge the adversary items to the marked items, and
(2) we charge the marked items to the algorithm's items.

One reason for doing it this way is that the adversary's ordering of the collected items (from smallest to largest) is more or less the reversed marking ordering (from largest to smallest). So the first step takes care of this "reversal" effect, while in the second step we focus on how we can pay for the marked items.

When we mark an item h, we collect an item i with $w_i \geq w_h/\phi$, so its budget ϕw_i is sufficient to pay for h. In a simple scenario, if we collect items in all steps, we can afford to pay for all marked items in step (2). In step (1), we also show that the weight of the marked items exceeds that of the adversary, and these two facts easily imply ϕ-competitiveness.

In reality, the situation is more complicated, since in some moves Algorithm MarkAndPick is idle, that is, it does not have any items to collect. For instance, suppose that the adversary collects an item j' in such a step. This item was collected by the algorithm in the past and is now marked. Its budget is $\phi w_{j'}$, and it pays for its mark, as well as for one other item collected by the adversary in the past. Roughly, this is the item collected by the adversary when the algorithm was marking j'.

The principle of this is as follows. Any idle step is a consequence of a step in which the algorithm marked $h = j'$, collected an item i, and the adversary collected an item j smaller than i and still pending for the algorithm. (This is not exactly correct, but it reflects the main principle. It may happen that the marked item h "responsible" for the idle step in which the adversary collects j' is actually different from j', but later in the process h "transfers" this responsibility to j'.) In that case, $w_j \leq w_{j'}/\phi$. We refer to such j's as "extra" items (even though those are not exactly the extra moves, but they cause extra moves later).

To organize the accounting so that we can pay for these extra items, we do two things. One, we do not immediately give MarkAndPick credit for the collected items. We give it credit for collecting i only at the time when the adversary collects an item that is at least as large as i. This is formalized below in Lemma 9 where the collected items i that are greater than the maximum adversary item contribute to the both sides of invariant (a) (they get included in L_t and L'_t), and only when the adversary collects an item greater than i, item i will be removed from L'_t and contribute to preserving the invariant.

Next, we keep track of the adversary extra items so that we can pay for them later. When the adversary collects an extra item j and we mark h, we know that

$w_j \leq w_h/\phi$. We store j in a separate set E' and h in M'. Later, when there is an idle step at which the adversary collects h, since h has already been collected by the algorithm, its budget ϕw_h pays both for w_h (for the mark on h) and for w_j. Once the adversary "consumed" the extra step when collecting h, we move j to a set E – the extra adversary items that we already paid for.

We represent our invariants in terms of the domination relations between some varying sets of items. The reason for this is that items can be added and removed from these sets in this process, and it does not seem possible to maintain appropriate bounds only between the total weights of these sets.

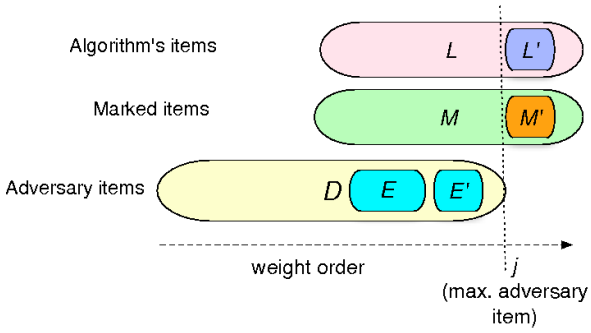

Fig. 1. Notation.

Notation: Symbols D_t, M_t, and L_t represent respectively the sets of items collected by the adversary, marked by the algorithm, and collected by Algorithm MarkAndPick up to and including step t. $L'_t = L_t \cap C_t$ is the set of algorithm's items that the adversary may still collect in the future. Let also $e_t = |D_t| - |M_t|$ and $\ell_t = |L'_t|$. Figure 1 illustrates this notation, as well as the sets introduced in the lemma below.

Lemma 9. *For each time step t, there exist disjoint sets $E_t, E'_t \subseteq D_t$ with $|E_t| = e_t$ and $|E'_t| = \ell_t$, and a set $M'_t \subseteq M_t$ with $|M'_t| = \ell_t$, such that*

(a) $\phi w(L_t - L'_t) \geq w(M_t - M'_t) + w(E_t)$,
(b) $\phi L'_t \succeq M'_t \succeq L'_t$,
(c) $M_t \succeq (D_t - E_t - E'_t) \cup L'_t$, *and*
(d) $M'_t \succeq \phi E'_t$.

Proof. We show that the invariant in the lemma is preserved. For simplicity, we omit the subscript t and write $D = D_t$, $M = M_t$, etc. Also, let $\Delta w(D) = w(D_{t+1}) - w(D_t)$, $\Delta w(M) = w(M_{t+1}) - w(M_t)$, and so on.

We view the process as follows: at each step,

(I) The adversary first inserts items into $\mathcal{S}$,

(II) then he selects the item j to be collected,
(III) next, the adversary deletes some items from $\mathcal{S}$ (of course, only the items that are before j in $\mathcal{S}$ can be deleted),
(IV) finally, both the adversary and the algorithm collect their items.

In order to show (a), we need to show that

$$\phi\Delta w(L) + \Delta w(M') \geq \phi\Delta w(L') + \Delta w(M) + \Delta w(E). \tag{3}$$

In addition, we need to show that (b), (c) and (d) are preserved.

We look at all sub-steps separately. (I) Insertions do not affect the invariants. (None of the sets M, D, L, L' changes, and we do not change sets M', E, and E'.) In (II), suppose the adversary selects j and j' was an item collected by the adversary in the previous step. There may have been some items i, $j' \lhd i \lhd j$, that were in L'. Since these items will not be in C in the next step, they will be removed from L', and we also need to update M' and E' so that they have the same cardinality as L', and in such a way that the invariants are preserved. Let $i \in L'$ be such an item with minimum weight, g the minimum-weight item in M' and e the minimum-weight item in E'. We remove i from L', g from M' and e from E'. Since $\phi w_i \geq w_g$, by (b), we have $\phi\Delta w(L) + \Delta w(M') = 0 - w_g \geq -\phi w_i + 0 + 0 = \phi\Delta w(L') + \Delta w(M) + \Delta w(E)$, so (a) is preserved. Invariants (b) and (d) are preserved because we remove the minimum items from L', M', and E'. In (c), the left-hand side does not change and on the right-hand side we remove i from L' and add e to $D - E - E'$, and by (b) and (d) we have $w_i \geq w_g/\phi \geq w_e$, so the right-hand side cannot increase. In sub-step (III), deletions do not affect the invariants.

The rest of the proof is devoted to sub-step (IV). We examine (3) and the changes in (b), (c) and (d) due to the algorithm and the adversary collecting their items.

Case A: There is at least one pending item. The algorithm marks h and collects the earliest pending item i such that $w_i \geq w_h/\phi$. Thus h is added to M and i is added to L. We do not change E. We have some sub-cases.

Case A.1: $i \trianglelefteq j$. Then i is not added to L', and we do not change E'. Since $\phi\Delta w(L) = \phi w_i \geq w_h = \Delta w(M)$ and $\Delta w(E) = 0$, to prove (3) it is now sufficient to show that

$$\Delta w(M') \geq \phi\Delta w(L'). \tag{4}$$

If $j \notin L$ (note that this included the case $i = j$), then L' does not change and we do not change M', so (4) is trivial. In (b) and (d) nothing changes. We add the maximum unmarked item h to M and j to D, so (c) follows from Lemma 8(iii).

If $i \neq j$ and $j \in L$ and then let g be the minimum item in M' and e the minimum item in E'. Item j is removed from L' and we remove g from M' and e from E'. Since, by (b), $\phi w_j \geq w_g$, we have $\Delta w(M') = -w_g \geq -\phi w_j = \phi\Delta w(L')$, so (4) holds. Since j, g and e are minimal, (b) and (d) are preserved. In (c), moving j from L' to D does not change the right-hand side. We also add h to

M and e to $D - E - E' \cup L'$, so (c) is preserved because of the choice of h and Lemma 8(iii).

Case A.2: $i \rhd j$ and $j \notin L$. Then i is added to L'. We also add j to E'. Since $\Delta w(L) = \Delta w(L') = w_i$ and $\Delta w(M) = w_h$, to show (3) it is sufficient to show that

$$\Delta w(M') \geq w_h. \tag{5}$$

Since $|L'|$ increased, we need to add one item f to M'. We choose this f as follows: If $i \trianglelefteq h$, we choose $f = h$. Otherwise, if $i \rhd h$ then, by the choice of h, we get that all items $h \trianglelefteq f \trianglelefteq i$ are now marked. In this case, we choose the largest $f \trianglelefteq i$ such that $f \notin M'$ and add it to M'. Since $h \in M - M'$, h itself is a candidate for f, so we have $h \trianglelefteq f \trianglelefteq i$.

Note that, in this case, by the choice of i (as the earliest pending item with weight at least w_h/ϕ), $j \lhd i$ and $j \notin L$, we have $w_f \geq w_h \geq \phi w_j$. In particular, this means that $j \lhd h$ and that all items $h \trianglelefteq f' \lhd i$ are in L'.

Since $w_f \geq w_h$, (5) is trivial. Invariant (d) is also quite easy, since $w_f \geq \phi w_j$, by the previous paragraph. In (c), adding j to D and E' does not change the right-hand side. We also add h to M and i to L', which preserves (c) by the choice of h and Lemma 8(iii).

To show (b), if $i \trianglelefteq h$ then $f = h$ and, since $\phi w_i \geq w_h \geq w_i$, invariant (b) is preserved. If $i \rhd h$, then, $\phi w_i \geq w_i \geq w_f$, so the first part of (b) is preserved. That the second part of (b) is preserved follows from the choice of f and Lemma 8(iii).

Case A.3: $i \rhd j$ and $j \in L$. Then we remove j from L' and add i. We thus have $\Delta w(L) = w_i$, $\Delta w(L') = w_i - w_j$ and $\Delta w(M) = w_h$. Thus to show (3) it is sufficient to show that

$$\Delta w(M') + \phi w_j \geq w_h. \tag{6}$$

We do not change E'. To update M', we proceed as follows. Let g be the lightest item in M'. Since j is the minimal element of L', (b) implies $w_j \leq w_g \leq \phi w_j$. We first remove g from M'. Next, we proceed similarly as in the previous case, looking for an item f that we can add to M' to compensate for removing g (since $|M'|$ cannot change in this case.) Let $h' = \max(g, h)$. Note that $h' \in M - M'$. If $i \trianglelefteq h'$, we choose $f = h'$. Otherwise, if $i \rhd h'$ then, by the choice of h', we get that all active items $h' \trianglelefteq f \trianglelefteq i$ are marked. In this case, we choose the largest $f \trianglelefteq i$ such that $f \notin M'$ and add it to M'. Since $h' \in M - M'$, h' itself is a candidate for f, so we have $h' \trianglelefteq f \trianglelefteq i$.

Note that, in this case, by $j \trianglelefteq g$, all items $h' \trianglelefteq f \lhd i$ are active and, by the choice of i, they are all in L'.

Now, in (6) we have $\Delta w(M') + \phi w_j = (-w_g + w_f) + \phi w_j \geq w_f \geq w_{h'} \geq w_h$. In (d), the left-hand side can only increase (since $f \geq g$) and the right-hand side does not change. In (c), moving j from L' to D does not change the right-hand side. We also added h to the left-hand side and i to L' on the right-hand side, so (c) is preserved by the choice of h and Lemma 8(iii).

In (b), removing j from L' and g from M' does not affect the invariant. Then we add f to M' and i to L'. By the algorithm, we have $\phi w_i \geq w_h$, while by the

case assumption and (b), we have $\phi w_i \geq \phi w_j \geq w_g$. Therefore $\phi w_i \geq w_{h'}$. Since either $f = h'$ or $f \trianglelefteq i$, this implies $\phi w_i \geq w_f$, showing that the first inequality in (b) is preserved. The second part of (b) follows again from the choice of f and Lemma 8(iii).

Case B: There are no pending items for the algorithm. It means that L' contains all active items $i \trianglerighteq j$, including j. By the weight ordering assumption and the second part of (b) this implies that $L' = M'$. Since the adversary collects an item and the algorithm does not, $e = |D| - |M|$ increases by 1, so we also need to add an item to E. Let b be the minimum-weight item in E'. We do this: we remove j from M' and from L' and we move b from E' to E. Using the choice of b and (d) we have $w_j \geq \phi w_b$, so

$$\begin{aligned}\phi\Delta w(L) + \Delta w(M') \;&=\; 0 + (-w_j) \;=\; -\phi w_j + w_j/\phi \;\geq \\ &\geq\; -\phi w_j + 0 + w_b \;=\; \phi\Delta w(L') + \Delta w(M) + \Delta w(E),\end{aligned}$$

and thus (3) holds. By the choice of j and e as the minimum items in L' and E', respectively, invariants (b) and (d) are preserved. In (c), j moves from L' to D, and b moves from E' to E, so the right-hand side does not change. □

Now we are ready to prove the main result of this section.

Theorem 7. *Algorithm* MarkAndPick *is ϕ-competitive for dynamic queues if item weights are non-decreasing.*

Proof. From the invariants in Lemma 9, at each time step we have

$$\begin{aligned}\phi w(L_t) &\geq [\phi w(L'_t) - w(M'_t)] + w(M_t) + w(E_t) \\ &\geq 0 + [w(D_t - E_t - E'_t) + w(L'_t)] + w(E_t) \\ &= w(D_t) + w(L'_t) - w(E'_t) \\ &\geq w(D_t) + w(M'_t)/\phi - w(E'_t) \\ &\geq w(D_t),\end{aligned}$$

and the ϕ-competitiveness follows. □

5.6 An Upper Bound of ϕ for Instances with 3-Bounded Life Span

Define a *life span* of an item to be the difference between the times of its deletion and insertion, or the number of steps when the item is active. We now consider instances where the life span of all items is bounded by 3 and show a ϕ-competitive algorithm for this problem. This is an easy adaptation of the algorithm in [6] and is included here only for completeness.

Algorithm FirstNotTooLight*:* At any step, collect the earliest pending item f such that $w_f \geq w_h/\phi$, where w_h is the heaviest pending item.

Fact 6 *The competitive ratio of Algorithm* FirstNotTooLight *for instances with* 3*-bounded life-span is* ϕ*.*

Proof. The proof is by a charging scheme. We start with a basic charging scheme: If the adversary collects an item x at time t, and the algorithm collects x at time t or earlier, charge x to itself. Else, charge x to the item collected by the algorithm at time t. This scheme does not quite work, so we will modify it slightly in the course of the analysis.

Suppose that at some time t the algorithm collects an item a. If a gets only one charge, then either it is a charge from itself, which is at most w_a, or a charge from the item collected by the adversary at time t, and then this item is pending for the algorithm at time t, so this charge is at most ϕw_a.

Assume now that a gets two charges, from the time when the adversary collects a at time $t' > t$ and from x collected at time t. If $a = h$ (where h is the heaviest item at time t), then the inequality $w_x \leq w_a/\phi$ is trivial, by the algorithm, so the charge to a is at most $w_a + w_x \leq (1 + 1/\phi)w_a = \phi w_a$.

So we can assume $a \neq h$. Clearly, the items a, x are different. By the EEF property, we have $x \lhd a$, which implies that $w_x \leq w_h/\phi < w_h$, and thus x, a, h are three different items and $x \lhd a \lhd h$. The adversary collects x and a, and thus we can assume he also collects h (for otherwise, he could rather collect h than x, since $w_h > w_x$). As h is active at time t and its life-span is at most 3, the adversary must collect a at time $t + 1$ and h at time $t + 2$.

Let g be the item collected by the algorithm at step $t + 1$. We charge half of w_x to a and half to w_g. Since the algorithm did not collect x at time t, we have $w_a \geq w_x$, so the charge to a is at most $w_a + w_x/2 \leq 1.5w_a \leq \phi w_a$. Similarly, the charge to g (since it does not get a charge from a) is at most $w_g + w_x/2 \leq 1.5w_g \leq \phi w_g$.

Overall, we showed that we can charge all items collected by the adversary to items collected by the algorithm in such a way that each item a receives a charge of at most ϕw_a. Thus the total value collected by the adversary is at most ϕ times the value collected by Algorithm FirstNotTooLight. □

6 Dynamic Queue — Randomized Algorithms

In this section we consider randomized algorithms for dynamic queues. Chin *et al.* [6] designed a randomized algorithm called RMix for bounded-delay packet scheduling. This algorithm is memoryless and achieves competitive ratio $\mathrm{e}/(\mathrm{e}-1)$ against an adaptive adversary. The competitiveness proof for RMix applies, with virtually no changes, to dynamic queues. In this section we show that this bound is tight.

Theorem 8. *For dynamic queues, any memoryless randomized algorithm has a competitive ratio at least* $\frac{\mathrm{e}}{\mathrm{e}-1}$ *against an adaptive-online adversary.*

Proof. Fix some online memoryless randomized algorithm $\mathcal{A}$. Recall that by a memoryless algorithm we mean an algorithm that makes a decision on which item to collect based only on the weights of the pending items.

We consider the following scheme. Let $a > 1$ be a constant which we specify later and n be a fixed integer. At the beginning, the adversary inserts items $a^0, a^1, \ldots, a^n$. (To simplify notation, in this proof we identify items with their weights.) In our construction we assure that in each step, the list of items which are pending for $\mathcal{A}$ is equal to $(a^0, a^1, \ldots, a^n)$. Since $\mathcal{A}$ is memoryless, in each step it uses the same probability distribution $(q_j)_{j=0}^n$, where q_j is the probability of collecting item a^j. As the algorithm always makes a move, $\sum_{i=0}^n q_i = 1$.

We consider $n+1$ strategies for an adversary, numbered $0, 1, \ldots, n$. The k-th strategy is as follows: in each step collect a^k, delete all items $a^0, a^1, ..., a^k$, and then issue new copies of all these items. Additionally, if the algorithm collected a^j for some $j > k$, then the adversary issues a new copy of a^j as well. This way, in each step exactly one copy of each a^j is pending for the algorithm.

This step is repeated $T \gg n$ times, and after the last step the adversary collects all uncollected items. Since $T \gg n$, we only need to focus on the expected amortized profits in a single step.

We look at the gains of $\mathcal{A}$ and the adversary in a single step. If the adversary chooses strategy k, then it gains a^k. Additionally, at the end it collects the item collected by the algorithm if this item is greater than a^k. Thus, its amortized expected gain in single step is $a^k + \sum_{i>k} q_i a^i$. The expected gain of $\mathcal{A}$ is $\sum_i q_i a^i$.

For any probability distribution $(q_j)_{j=0}^n$ of the algorithm, the adversary chooses a strategy k which maximizes the competitive ratio. Thus, the competitive ratio of $\mathcal{A}$ is is at least

$$\mathcal{R} = \max_k \left\{ \frac{a^k + \sum_{j>k} q_j a^j}{\sum_j q_j a^j} \right\} \;\geq\; \sum_k v_k \frac{a^k + \sum_{j>k} q_j a^j}{\sum_j q_j a^j} \quad ,$$

for any coefficients $v_0, ..., v_n \geq 0$ such that $\sum_k v_k = 1$. Note that the latter term corresponds to the ratio forced by a randomized adversary who chooses k with probability v_k. In particular, we may choose v_k to be the value for which the competitive ratio of such a randomized adversary strategy against *any deterministic* algorithm is the same. After solving the set of equations we get

$$v_k = \begin{cases} \frac{1}{M} a^{n-k}(a-1) & \text{if } k < n \\ \frac{1}{M}\left(a - n(a-1)\right) & \text{if } k = n \end{cases} \qquad \text{where } \; M = a^{n+1} - n(a-1) \;\; .$$

For these values of v_k we get

$$\begin{aligned} M\mathcal{R} \textstyle\sum_j q_j a^j &\geq \sum_k M v_k a^k + \sum_k M v_k \sum_{j>k} q_j a^j \\ &= \sum_{k=0}^{n-1} M v_k a^k + M v_n a^n + \sum_j q_j a^j \sum_{k<j} M v_k \\ &= n(a-1)a^n + \left(a - n(a-1)\right) a^n + \sum_j q_j a^j \left(a^{n+1} - a^{n-j+1}\right) \\ &= a^{n+1} \sum_j q_j a^j \;\; . \end{aligned}$$

Therefore, $\mathcal{R} \geq \frac{a^{n+1}}{M}$. This bound is maximized when we choose $a = 1 + 1/n$, in which case

$$\mathcal{R} \;\geq\; \frac{\left(1+\frac{1}{n}\right)^{n+1}}{\left(1+\frac{1}{n}\right)^{n+1}-1} \;,$$

which is arbitrarily close to $\frac{\mathrm{e}}{\mathrm{e}-1}$ for large n. □

7 Final Comments

We provided some upper and lower bounds for the competitive ratio of several versions of the item selection problem. Many open problems remain. Although, as we show, for the general case there is a simple 2-competitive algorithm and no better ratio is possible, in most other versions of item collection gaps between the lower and upper bounds remain.

The most intriguing open problems are to establish better bounds for:

(i) *The general randomized case.* Here, we have an $\mathrm{e}/(\mathrm{e}-1)$ lower bound, and we show that it is tight in the uniform decremental case.

(ii) *The dynamic queue case.* For the queue, we showed a lower bound of ≈ 1.63 (improving the lower bound of ϕ), but no upper bound better than 2 is known. (Better bounds are known for packet scheduling [17, 8], but these algorithm use information about packet deadlines and do not seem to apply to dynamic queues.) We showed better bounds for some restricted cases: 1.8 for the FIFO queue (a generalization of packet scheduling with agreeable deadlines) and ≈ 1.737 for the decremental queue.

We showed several lower bounds for memoryless algorithms. In our definition, a memoryless algorithm can use information about the weight of the pending items. One could relax this definition and allow the algorithm to know the weights of all active packets, and we do not know whether similar bounds can be proved in this case.

References

1. N. Andelman, Y. Mansour, and A. Zhu. Competitive queueing policies in QoS switches. In *Proc. 14th Symp. on Discrete Algorithms (SODA)*, pages 761–770. ACM/SIAM, 2003.
2. Y. Azar and Y. Richter. An improved algorithm for CIOQ switches. In *Proc. 12th European Symp. on Algorithms (ESA)*, volume 3221 of *LNCS*, pages 65–76. Springer, 2004.
3. N. Bansal, L. Fleischer, T. Kimbrel, M. Mahdian, B. Schieber, and M. Sviridenko. Further improvements in competitive guarantees for QoS buffering. In *Proc. 31st International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 3142 of *LNCS*, pages 196–207. Springer, 2004.
4. Y. Bartal, F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, R. Lavi, J. Sgall, and T. Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. In *Proc. 21st Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 2996 of *LNCS*, pages 187–198. Springer, 2004.

5. J.-P. Bouillon, C. Portella, J. Bouquant, and S. Humbel. Theoretical study of intramolecular aldol condensation of 1,6-diketones trimethylsilyl substituent effect. *Journal of Organic Chemistry*, 65:5823–30, 2000.
6. F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, J. Sgall, and T. Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. *Journal of Discrete Algorithms*, 4:255–276, 2006.
7. F. Y. L. Chin and S. P. Y. Fung. Online scheduling for partial job values: Does timesharing or randomization help? *Algorithmica*, 37:149–164, 2003.
8. M. Englert and M. Westerman. Considering suppressed packets improves buffer management in QoS switches. In *Proc. 18th Symp. on Discrete Algorithms (SODA)*, pages 209–218. ACM/SIAM, 2007.
9. L. Epstein, J. Noga, and G. J. Woeginger. On-line scheduling of unit time jobs with rejection: minimizing the total completion time. *ORL*, 30(6):415–420, 2002.
10. S. Gutiérrez, S. O. Krumke, N. Megow, and T. Vredeveld. How to whack moles. *Theor. Comput. Sci.*, 361(2):329–341, 2006.
11. B. Hajek. On the competitiveness of online scheduling of unit-length packets with hard deadlines in slotted time. In *Conference in Information Sciences and Systems*, pages 434–438, 2001.
12. A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. In *Proc. 33rd Symp. Theory of Computing (STOC)*, pages 520–529. ACM, 2001.
13. A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. *SIAM J. Comput.*, 33:563–583, 2004.
14. A. Kesselman, Y. Mansour, and R. van Stee. Improved competitive guarantees for QoS buffering. In *Proc. 11th European Symp. on Algorithms (ESA)*, volume 2832 of *LNCS*, pages 361–372. Springer, 2003.
15. A. Kesselman, Y. Mansour, and R. van Stee. Improved competitive guarantees for QoS buffering. *Algorithmica*, 43:63–80, 2005.
16. F. Li, J. Sethuraman, and C. Stein. An optimal online algorithm for packet scheduling with agreeable deadlines. In *Proc. 16th Symp. on Discrete Algorithms (SODA)*, pages 801–802. ACM/SIAM, 2005.
17. F. Li, J. Sethuraman, and C. Stein. Better online buffer management. In *Proc. 18th Symp. on Discrete Algorithms (SODA)*, pages 199–208. ACM/SIAM, 2007.
18. A. C. C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. 18th Symp. Foundations of Computer Science (FOCS)*, pages 222–227. IEEE, 1977.