Improved Online Algorithms for Buffer Management in QoS Switches

Marek Chrobak[1], Wojciech Jawor[1], Jiří Sgall[2], and Tomáš Tichý[2]

[1] Department of Computer Science, University of California, Riverside, CA 92521.
marek,wojtek@cs.ucr.edu

[2] Mathematical Institute, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic.
sgall,tichy@math.cas.cz

Abstract. We consider the following buffer management problem arising in QoS networks: packets with specified weights and deadlines arrive at a network switch and need to be forwarded so that the total value of forwarded packets is maximized. If a packet is not forwarded before its deadline, it is lost and brings no profit. The main result of the paper is an online $\frac{64}{33} \approx 1.939$-competitive algorithm – the first deterministic algorithm for this problem with competitive ratio below 2. We also study the *s-uniform* case, in which all packets have the same span s (the difference between the deadline and the arrival time), for which we give an algorithm with ratio $5 - \sqrt{10} \approx 1.838$. Our algorithm achieves the same ratio in a more general scenario when all packets are similarly ordered. Our last two results concern the 2-uniform case, for which we give an algorithm with ratio ≈ 1.377 and a matching lower bound.

1 Introduction

One of the issues arising in IP-based QoS networks is how to manage the packet flows at a router level. In particular, in case of overloading, when the total incoming traffic exceeds the buffer size, the buffer management policy needs to determine which packets should be dropped by the router. Kesselman *et al.* [5] postulate that the packet drop policies can be modeled as combinatorial optimization problems. Of the two models proposed in [5], the one relevant to this work is called *buffer management with bounded delay*, and is defined as follows: Packets arrive at a network switch. Each packet is characterized by a positive weight and a deadline before which it must be transmitted. Packets can only be transmitted at integer time steps. If the deadline of a packet is reached while it is still being buffered, the packet is lost. The goal is to maximize the weighted number of forwarded packets.

It is easy to see that this buffer management problem is equivalent to the online version of the following unit job scheduling problem. We are given a set of unit-length jobs, with each job j specified by a triple (r_j, d_j, w_j), where r_j and d_j are integral release times and deadlines, and w_j is a non-negative real weight. One job can be processed at each integer time. We use the term *weighted* *throughput* or *gain* for the total weight of the jobs completed by their deadline. The goal is to compute a schedule that maximizes the weighted throughput.

In the online version of the problem jobs arrive in time and the algorithm needs to schedule one of the pending jobs without knowledge of the future. An online algorithm $\mathcal{A}$ is *R-competitive* if its gain on any instance is at least $1/R$ times the optimal (offline) gain on this instance. The *competitive ratio* of $\mathcal{A}$ is the infimum of such values R. It is common in the literature to view the behavior of an online algorithm $\mathcal{A}$ as a game between $\mathcal{A}$ and an *adversary*, who issues the jobs and schedules them in order to maximize the ratio between his gain and and the gain of $\mathcal{A}$.

Past work. A simple greedy algorithm that always schedules the heaviest available job is 2-competitive, and this is the best previous bound for deterministic algorithms for this problem. A lower bound of $\phi \approx 1.618$ was shown in $[1, 3, 4]$.

Some restrictions on instances of the problem have been studied in the literature $[5, 1, 3]$. Let the span of a job be the difference between its deadline and the release time. In *s-uniform instances* the span of each job is equal exactly s. In *s-bounded* instances, the span of each job is at most s. The lower bound of $\phi \approx 1.618$ in $[1, 3, 4]$ applies even to 2-bounded instances. A matching upper bound for the 2-bounded case was presented in [5]. Algorithms for 2-uniform instances were studied by Andelman *et al.* [1], who established a lower bound of $\frac{1}{2}(\sqrt{3}+1) \approx 1.366$ and an upper bound of $\sqrt{2} \approx 1.414$. This upper bound is tight for memoryless algorithms [2], that is, algorithms which base their decisions only on the weights of pending jobs and are invariant under scaling of weights. Finally, the first deterministic algorithms with competitive ratio lower than 2 for the s-bounded instances appear in [2]. These ratios, however, depend on s, and approach 2 as s increases. Kesselman *et al.* [6] gave an algorithm for s-uniform instances with ratio 1.983, independent of s.

A randomized 1.582-competitive algorithm for the general case was given in [2]. For 2-bounded instances, there is a 1.25-competitive algorithm [2] that matches the lower bound [3]. For the 2-uniform case the currently best lower bound for randomized algorithms is 1.172 [2].

Our results. We present several deterministic online algorithms for the buffer management problem. Our main result is a deterministic $\frac{64}{33} \approx 1.939$-competitive algorithm for the general case. This is the first deterministic algorithm for the problem with competitive ratio strictly below 2. All of the algorithms given previously in $[5, 1, 2, 6]$ achieved competitive ratio strictly below 2 only when additional restrictions on instances were placed.

For the s-uniform case, we give an algorithm with ratio $5 - \sqrt{10} \approx 1.838$. In fact, this result holds in a more general scenario when all jobs are similarly ordered, that is, when $r_i < r_j$ implies $d_i \leq d_j$ for all jobs i, j. Thus, compared to the 1.983-competitive algorithm given in [6], our algorithm achieves a better competitive ratio and works in a more general setting. Finally, we completely solve the 2-uniform case: we give an algorithm with competitive ratio ≈ 1.377 and a matching lower bound. Note that this ratio is strictly in-between the previous lower and upper bounds from [1].

2 Terminology and Notation

A *schedule* S specifies which jobs are executed, and for each executed job j it specifies an integral time t, $r_j \le t < d_j$, when it is scheduled. Only one job can be scheduled at any t. The *throughput* or *gain* of a schedule S is the total weight of the jobs executed in S. If $\mathcal{A}$ is a scheduling algorithm, by $gain_{\mathcal{A}}(I)$ we denote the gain of the schedule computed by $\mathcal{A}$ on I. The optimal gain on I is denoted by $opt(I)$. A job i is *pending* in S at time t if $r_i \le t < d_i$ and i has not been scheduled before t. (Thus all jobs released at time t are considered pending.)

An instance is *s-bounded* if $d_j - r_j \le s$ for all jobs j. Similarly, an instance is *s-uniform* if $d_j - r_j = s$ for all j. The difference $d_j - r_j$ is called the *span* of a job j. An instance is called *similarly ordered* when the release times and deadlines are similarly ordered, that is if $r_i < r_j$ implies $d_i \le d_j$ for any two jobs i and j.

Given two jobs i, j, we say that i *dominates* j if either (i) $d_i < d_j$, or (ii) $d_i = d_j$ and $w_i > w_j$, or (iii) $d_i = d_j$, $w_i = w_j$ and $i < j$. (Condition (iii) only ensures that ties are broken in some arbitrary but consistent way.) Given a non-empty set of jobs J, the *dominant* job in J is the one that dominates all other jobs in J; it is always uniquely defined as 'dominates' is a linear order.

A schedule S is called *canonical earliest-deadline* if for any jobs i and j in S, where i is scheduled at time t and j scheduled later, either j is released strictly after time t, or i dominates j. In other words, at any time, the job to be scheduled dominates all pending jobs that appear later in S. Any schedule can be easily converted into a canonical earliest-deadline schedule by rearranging its jobs. Thus we may assume that offline schedules are canonical earliest-deadline.

3 A $\frac{64}{33}$-Competitive Algorithm

We start with some intuitions that should be helpful in understanding the algorithm and its analysis. The greedy algorithm that always executes the heaviest job (H-job) is not better than 2-competitive. An alternative idea is to execute the earliest deadline job at each step. This algorithm is not competitive at all, for it could execute many small weight jobs even if there are heavy jobs pending with only slightly larger deadlines. A natural refinement of this approach is to focus on sufficiently heavy jobs, of weight at least α times the maximal weight, and chose the earliest deadline job among those (an E-job). As it turns out, this algorithm is also not better than 2-competitive.

The general idea of our new algorithm is to alternate H-jobs and E-jobs. Although this simple algorithm, as stated, still has ratio no better than 2, we can reduce the ratio by intrudocing some minor modifications.

Algorithm GenFlag**:** We use parameters $\alpha = \frac{7}{11}$, $\beta = \frac{8}{11}$, and a boolean variable *eflag*, initially equal *false*, that stores information about the previous step. At a given time step t, update the set of pending jobs (remove jobs with deadline t and add jobs released at t). If there are no pending jobs, go to the next time step. Otherwise, let h be the heaviest pending job (breaking ties in favor of dominant jobs) and e the dominant job among the pending jobs with weight at least αw_h. Schedule either e or h according to the following procedure:

```
if eflag = false then
        schedule e
        if e ≠ h then set eflag ← true
else
        if d_e = t + 1 and w_e ≥ βw_h then schedule e
        else   schedule h
        set eflag ← false
```

A job e scheduled while *eflag*=*false* is called an *O-job* if $e = h$, or an *E-job* otherwise. A job e scheduled while *eflag*=*true* is called an *U-job*. A job h scheduled in the last case is called an *H-job*. The letters stand for **O**bvious, **E**arly, **U**rgent, and **H**eaviest.

Variable *eflag* is *true* iff the previous job was an E-job. Thus, in terms of the labels, the algorithm proceeds as follows: If an O-job is available, we execute it. Otherwise, we execute an E-job, and in the next step either a U-job (if available) or an H-job. (There will always be a pending job at this next step, thanks to the condition $e \neq h$: if $d_h = t$ then $e = h$ by the definition of dominance; so if an E-job is executed at time t, $d_h > t$ and h will be pending in the next step.)

Theorem 1. GenFlag *is a* $\frac{64}{33}$*-competitive deterministic algorithm for unit-job scheduling.*

Proof. The proof is by a charging scheme. Fix an arbitrary (offline) schedule ADV. For each job j executed in ADV, we partition its weight w_j into several *charges*, and assign each charge to a job executed by GenFlag. If the total charge to each job i of GenFlag were at most Rw_i, the R-competitiveness of GenFlag would follow by summation over all jobs. Our charging scheme does not always meet this simple condition. Instead, we divide the jobs of GenFlag into disjoint groups, where each group is either a single O-job, or an EH-pair (an E-job followed by an H-job), or an EU-pair (an E-job followed by a U-job). This is possible by the discussion of types of jobs before the theorem. For each group we prove that its *charging ratio* is at most R, where the charging ratio is defined as the total charge to this group divided by the total weight of the group. This implies that GenFlag is R-competitive by summation over all groups.

Charging scheme. Let j be the job executed at time t in ADV. Denote by i and h, respectively, the job executed by GenFlag and the heaviest pending job at time t. (If there are no pending jobs, introduce two "dummy" jobs of weight 0. This does not change the algorithm.) Then j is charged to GenFlag's jobs, according to the following rules.

(EB) If j is executed by GenFlag before time t, then charge $(1-\beta)w_h$ to i and the remaining $w_j - (1-\beta)w_h$ to j.

(EF) Else, if j is executed by GenFlag after time t, then charge βw_j to i and $(1-\beta)w_j$ to j.

(NF) Else, if i is an H-job, $w_j \geq \beta w_i$, and ADV executes i after time t, then charge βw_j to i and $(1-\beta)w_j$ to the job scheduled by GenFlag at time $t+1$.

(U) Else, charge w_j to i. (Note that this case includes the case $i = j$.)

We label all charges as EB, EF, NF, U, according to which case above applies. We also distinguish upward, forward, and backward charges, defined in the obvious way. Thus, for example, in case (EB), the charge of $w_j - (1-\beta)w_h$ to j is a backward EB-charge. The letters in the labels refer to whether j was executed by GenFlag, and to the charge direction: **E**xecuted-**B**ackward, **E**xecuted-**F**orward, **N**on-executed-**F**orward, **U**pward. We start with several simple observations that will be used later in the proof, sometimes without explicit reference.

Observation 1: At time t, let h be the heaviest job of GenFlag and e the dominant job of weight at least αw_h. Let j be any pending job. Then

(1.1) $w_j \leq w_h$.

(1.2) If j dominates e then $w_j < \alpha w_h$.

Proof: Inequality (1.1) is trivial, by the definition of h. Inequality (1.2) follows from the fact that e dominates all pending jobs with weight at least αw_h.

Observation 2: Suppose that at time t GenFlag schedules a job that receives a forward NF-charge from the job l scheduled at time $t-1$ in ADV. Then l is pending at time t.

Proof: Denote by f the heaviest pending job of GenFlag at time $t-1$. By the definition of NF-charges, $l \neq f$, l is pending at time $t-1$, f is executed as an H-job, $d_f \geq t+1$, and $w_l \geq \beta w_f$. Therefore $d_l \geq t+1$, since otherwise GenFlag would execute l (or some other job) as a U-job at step $t-1$. Thus l is also pending at step t, as claimed.

Observation 3: Suppose that at time t GenFlag schedules a job e of type O or E, and h is the heaviest pending job at time t. Then

(3.1) The upward charge to e is at most w_h.

(3.2) If ADV executes e after time t then the upward charge is at most αw_h.

(3.3) The forward NF-charge to e is at most $(1-\beta)w_h$.

Proof: Denote by j the job executed at time t in ADV. If j is scheduled by GenFlag before time t, then the upward charge is $(1-\beta)w_h \leq \alpha w_h$ and claims (3.1) and (3.2) hold. Otherwise j is pending at time t. Then the upward charge is at most $w_j \leq w_h$ and (3.1) follows. If ADV executes e after time t, then, since ADV is canonical, job j must dominate e, and (3.2) follows from (1.2). (3.3) follows by Observation 2 and (1.1).

Observation 4: Suppose that at time t GenFlag executes an H-job h that is executed after time t in ADV. Then the upward charge to h is at most βw_h.

Proof: Let j be the job executed at time t in ADV. In case (EB), the upward charge to h is at most $(1-\beta)w_h \leq \beta w_h$. In all other cases, j is pending at time t, so $w_j \leq w_h$. In cases (EF) and (NF), the charge is $\beta w_j \leq \beta w_h$. In case (U) the charge is w_j, but since (NF) did not apply, this case can occur only if $w_j \leq \beta w_h$.

This completes the proofs of all observations. Now we examine the charges to all job groups in GenFlag's schedule: single O-jobs, EH-pairs, and EU-pairs.

O-jobs. Let $e = h$ be an O-job executed at time t. The forward NF-charge is at most $(1-\beta)w_e$. If e gets a backward EB-charge then e gets no forward EF-charge and the upward charge is at most αw_e; the total charging ratio is at most $2+\alpha-\beta < R$. If e does not get a backward EB-charge then the forward EF-charge is at most $(1-\beta)w_e$ and the upward charge is at most w_e; the charging ratio is at most $3-2\beta < R$.

E-jobs. Before considering EH-pairs and EU-pairs, we estimate separately the charges to E-jobs. Suppose that at time t GenFlag executes an E-job e, and the heaviest job is h. We claim that e is charged at most $\alpha w_h + (2-\beta)w_e$. We have two cases.

<u>Case 1</u>: e gets no backward EB-charge. Then, in the worst case, e gets an upward charge of w_h, a forward NF-charge of $(1-\beta)w_h$, and a forward EF-charge of $(1-\beta)w_e$. Using $(2-\beta)w_h = 2\alpha w_h$ and $\alpha w_h \le w_e$, the total charge is at most $(2-\beta)w_h + (1-\beta)w_e \le \alpha w_h + (2-\beta)w_e$ as claimed.

<u>Case 2</u>: e gets a backward EB-charge. Then there is no forward EF-charge and the upward charge is at most αw_h by (3.2). Let l be the job scheduled at time $t-1$ in ADV. If there is an NF-charge, it is generated by l and then l is pending for GenFlag at time t by Observation 2.

If there is a forward NF-charge and $d_l \ge d_e$, then l is pending at the time when ADV schedules e. (In this case l cannot be executed by GenFlag, because charges EB, EF did not apply to l.) Consequently, e receives at most a backward EB-charge $w_e - (1-\beta)w_l$, a forward NF-charge $(1-\beta)w_l$, and an upward charge αw_h. The total is at most $\alpha w_h + w_e \le \alpha w_h + (2-\beta)w_e$, as claimed.

Otherwise, there is no forward NF-charge, or there is a forward NF-charge and $d_l < d_e$. In the second case, l is pending at t, and l dominates e, so the forward NF-charge is at most $(1-\beta)w_l \le (1-\beta)w_e$. With the backward EB-charge of w_e and upward charge of at most αw_h, the total is at most $\alpha w_h + (2-\beta)w_e$, as claimed.

EH-pairs. Let e be the E-job scheduled at time t, h the heaviest pending job at time t, and h' the H-job at time $t+1$. By the algorithm, $e \ne h$. Note that, since GenFlag did not execute h as a O-job at time t, h is still pending after the E-step and $w_{h'} \ge w_h$.

We now estimate the charge to h'. There is no forward NF-charge, as the previous step is not an H-step. If there is a backward EB-charge, the additional upward charge is at most $\beta w_{h'}$ and the total is at most $(1+\beta)w_{h'}$. If there is no EB-charge, the sum of the upward charge and a forward EF-charge is at most $w_{h'} + (1-\beta)w_{h'} \le (1+\beta)w_{h'}$. With the charge to e of at most $\alpha w_h + (2-\beta)w_e$, the total charge of the EH-pair is at most $\alpha w_h + (2-\beta)w_e + (1+\beta)w_{h'}$, and thus the charging ratio is at most

$$2-\beta+\frac{\alpha w_h + (2\beta-1)w_{h'}}{w_e + w_{h'}} \le 2-\beta+\frac{\alpha w_h + (2\beta-1)w_{h'}}{\alpha w_h + w_{h'}} \le 2-\beta+\frac{\alpha+2\beta-1}{\alpha+1} = R.$$

The first step follows from $w_e \ge \alpha w_h$. As $2\beta - 1 < 1$, the next expression is decreasing in $w_{h'}$, so the maximum is at $w_{h'} = w_h$.

EU-pairs. As in the previous case, let e, and h denote the E-job scheduled at time t and the heaviest pending job at time t. By g and h' we denote the scheduled U-job and the heaviest pending job at time $t+1$. As in the case of EH-pairs, $e \neq h$ and $w_{h'} \geq w_h$.

Job g gets no backward EB-charge, since it expires, and no forward NF-charge, since the previous step is not an H-step. The upward charge is at most $w_{h'}$, the forward EF-charge is at most $(1-\beta)w_g$.

With the charge to e of at most $\alpha w_h + (2-\beta)w_e$, the total charge of the EU-pair is at most $\alpha w_h + (2-\beta)w_e + w_{h'} + (1-\beta)w_g$, so the charging ratio is at most

$$2-\beta+\frac{\alpha w_h + w_{h'} - w_g}{w_e + w_g} \leq 2-\beta+\frac{\alpha w_h + (1-\beta)w_{h'}}{\alpha w_h + \beta w_{h'}} \leq 2-\beta+\frac{\alpha+1-\beta}{\alpha+\beta} = R.$$

In the first step, we apply bounds $w_e \geq \alpha w_h$ and $w_g \geq \beta w_{h'}$. As $1-\beta < \beta$, the next expression is decreasing in $w_{h'}$, so the maximum is at $w_{h'} = w_h$.

Summarizing, we now have proved that the charging ratio to all job groups is at most R, and the R-competitiveness of GenFlag follows.

4 A $(5-\sqrt{10})$-Competitive Algorithm for Similarly Ordered Jobs

We now consider the case when the jobs are similarly ordered, that is $r_i < r_j$ implies $d_i \leq d_j$ for all jobs i, j. Note that, in particular, this covers the case when all jobs on input have the same span.

Algorithm SimFlag**:** We use one parameter $\alpha = \sqrt{10}/5 \approx 0.633$ and a boolean variable *eflag*, initially set to *false*. At a given time step t, update the set of pending jobs (remove jobs with deadline t and add jobs released at t). If there are no pending jobs, go to the next time step. Otherwise, let h be the heaviest pending job (breaking ties in favor of dominant jobs) and e the dominant job among the pending jobs with weight at least αw_h. Schedule either e or h according to the following procedure:

```
if eflag = false then
    schedule e
    if e ≠ h  ∧  d_e > t+1  then set eflag ← true
else
    schedule h
    set eflag ← false
```

A job e scheduled while *eflag*=*false* is called an *O-job* if $e = h$ or $d_e = t+1$, and an *E-job* otherwise. A job h scheduled while *eflag*=*true* is called an *H-job*. The intuition behind the algorithm is similar to GenFlag, and, in fact, it is simpler. It schedules O-jobs, if available, and if not, it schedules an E-job followed by an H-job. (By condition $e \neq h$, there will be a pending job in the next step.)

Theorem 2. SimFlag *is a* $5-\sqrt{10} \approx 1.838$*-competitive deterministic algorithm for unit-job scheduling for similarly ordered instances.*

Proof. The proof is by a charging scheme. The steps of the algorithm can be divided into single O-steps, and EH-pairs. This is because each E-job must be followed by an H-job, and each H-job must be preceded by an E-job. Similarly as for GenFlag, we give a charging scheme and show that the charging ratio to each group is at most R.

Charging scheme. Let j be the job executed at time t in ADV. Denote by i and h, respectively, the job executed by SimFlag and the heaviest pending job at time t. (Without loss of generality, we can assume that such jobs exist.) Let $\beta = 4 - \sqrt{10} \approx 0.838$. Then j is charged to SimFlag's jobs, according to the following rules.

(EB) If j is executed by SimFlag at or before time t, then charge w_j to j.

(NF) Else, if $d_j > t+1$ and i is an H-job, then charge βw_j to i and $(1-\beta)w_j$ to the job scheduled by SimFlag at time $t+1$.

(U) Else, charge w_j to i.

The definition of canonical schedules and the assumption of similar ordering implies the following useful property: if ADV schedules j' before j then $d_{j'} \leq d_j$. We make a few more observations.

Observation 1: Consider the execution of SimFlag. At time t, let h be the heaviest job and e the dominant job of weight at least αw_h. Let j be any pending job. Then

(1.1) $w_j \leq w_h$.

(1.2) If j dominates e then $w_j < \alpha w_h$.

Observation 2: Suppose that at time t SimFlag schedules a job e (of type O or E), and h is the heaviest pending job at time t. Then

(2.1) The upward charge to e is at most w_h.

(2.2) If ADV executes e after time t then the upward charge is at most αw_h.

(2.3) The forward NF-charge to e is at most $(1-\beta)w_h$.

(2.4) If ADV executes e after time t then its forward NF-charge is at most $(1-\beta)w_e$.

(2.5) If ADV does not execute e after time t then the charge to e is at most Rw_e.

The proofs of (1.1)-(2.3) are the same as for GenFlag.

For (2.4), let l be the job executed at time $t-1$ in ADV; since e is scheduled later we have $d_l \leq d_e$. By the definition of NF-charges, l is pending at time t. This, by the choice of e, implies that $w_l \leq w_e$ and (2.4) follows.

In (2.5), e does not get a backward EB-charge, the upward charge is at most w_h and the forward NF-charge is at most $(1-\beta)w_h$. The total is at most $(2-\beta)w_h \leq (2-\beta)w_e/\alpha = Rw_e$.

O-jobs. Let e be an O-job executed at time t, and let h be the heaviest pending job. By the algorithm, either $e = h$ or $d_e = t+1$ (i.e., e expires). If e gets no backward charge, the charging ratio is at most R by (2.5). If e gets a backward EB-charge then $d_e > t+1$ and thus $e = h$. The upward charge is at most

$\alpha w_h = \alpha w_e$ and the forward NF-charge is at most $(1-\beta)w_e$. So the charging ratio is at most $2+\alpha-\beta < R$.

EH-pairs. Let e be the E-job scheduled at time t, h be the heaviest pending job at time t, and h' be the H-job at time $t+1$. Let j and j' be the jobs executed at times t and $t+1$ in ADV, respectively. By the algorithm, we have $e \neq h$, which implies $d_h > d_e$ as otherwise h is chosen as e. Furthermore, $d_e \geq t+2$ and $w_{h'} \geq w_h$.

Case 1: ADV schedules e after time t. Then j' dominates e or $j' = e$, as ADV schedules j' at $t+1$ when e is pending. Also, j dominates e. Thus the U-charges to e and h' are each at most αw_h. In addition, e could receive an NF-charge of at most $(1-\beta)w_e$ and the EB-charges are $w_e + w_{h'}$. The total charge to the EH-pair is at most $(2-\beta)w_e + 2\alpha w_h + w_{h'}$, and the charging ratio is at most

$$1 + \frac{(1-\beta)w_e + 2\alpha w_h}{w_e + w_{h'}} \leq 1 + \frac{(1-\beta)w_e + 2\alpha w_h}{w_e + w_h} \leq 1 + \frac{(1-\beta)\alpha + 2\alpha}{\alpha+1} = R.$$

The first inequality follows from $w_{h'} \geq w_h$, and the next inequality holds because its left-hand side is decreasing in $w_e \geq \alpha w_h$ (note that $1-\beta < 2\alpha$).

Case 2: ADV does not schedule e after time t. We estimate the charges to e and h' separately. The charging ratio of e is at most R by (2.5).

We claim that the upward charge to h' is at most $\beta w_{h'}$. If j' is scheduled before t, then there is no charge. If $d_{j'} > t+2$, then this follows by the NF-charge definition. Otherwise $d_{j'} \leq d_e < d_h$ and thus $r_{j'} \leq r_h \leq t$ and j' is pending at t. Thus $w_{j'} \leq w_e$, as otherwise j' would be chosen as e at time t (note that $e \neq j'$ by the case condition.) So the upward charge is at most $w_e \leq \alpha w_{h'} < \beta w_{h'}$ and the claim is proven. The total charge to h', including a possible EB-charge, is at most $(1+\beta)w_{h'} = R w_{h'}$ and the charging ratio is at most R.

5 2-Uniform Instances

In this section we consider 2-uniform instances, where each job j satisfies $d_j = r_j + 2$. Let $Q \approx 1.377$ be the largest root of $Q^3 + Q^2 - 4Q + 1 = 0$. First, we prove that no online algorithm for this problem can be better than Q-competitive. Next, we show that this lower bound is in fact tight.

5.1 Lower Bound

Fix some $0 < \epsilon < 2Q - 2$. We define a sequence Ψ_i, $i = 1, 2, \ldots$, as follows. For $i = 1$, $\Psi_1 = Q - 1 - \epsilon$. Inductively, for $i \geq 1$, let

$$\Psi_{i+1} = \frac{Q(2-Q)\Psi_i + 3Q^2 - 5Q + 1}{Q(2 - Q - \Psi_i)} = \frac{(2-Q)\Psi_i - (Q-1)^2}{2 - Q - \Psi_i},$$

where the second equality follows from $Q^3 + Q^2 - 4Q + 1 = 0$.

Lemma 1. *For all i, we have $1 - Q \leq \Psi_i < Q - 1$. Furthermore, the sequence $\{\Psi_i\}$ converges to $1 - Q$.*

Proof. Substituting $\Psi_i = z_i + 1 - Q$, we get $z_{i+1} = \frac{(3-2Q)z_i}{1-z_i}$. If $0 \leq z_i \leq 2Q - 2 - \epsilon$, then $0 \leq z_{i+1} < \frac{3-2Q}{3-2Q+\epsilon} z_i$. Thus $0 \leq z_i < 2Q - 2 - \epsilon$ for all i, $\lim_{i\to\infty} z_i = 0$, and the lemma follows.

Theorem 3. *There is no deterministic online algorithm for the 2-uniform case with competitive ratio smaller than Q.*

Proof. The proof is by contradiction. Suppose that $\mathcal{A}$ is a $(Q - \epsilon)$-competitive algorithm, for some $\epsilon > 0$. We develop an adversary strategy that will force $\mathcal{A}$'s ratio to be bigger than $Q - \epsilon$.

Throughout this proof, to simplify notation, we will identify jobs by their weight. Thus, when we say "job x", we mean the job with weight x. Such job will be always either uniquely defined by the context, or there will be two such identical jobs, in which case one of them can be chosen arbitrarily. At each step t, we distinguish *old pending jobs*, that is, those that were released at time $t - 1$ but not executed, from the newly released jobs. Without loss of generality, we can assume that there is always (except for the last step) exactly one old pending job. (All old pending jobs except the heaviest one can be ignored. If there are no no old pending jobs, add a "dummy" job one with weight 0.)

By a *configuration* we mean a pair $\langle x, y\rangle$, where x, y denote the old pending jobs of $\mathcal{A}$ and the adversary, respectively.

Let $\{\Psi_i\}$ be the sequence from Lemma 1. For $i \geq 1$ define

$$a_i = \frac{1-\Psi_i}{Q-1} \quad \text{and} \quad b_i = \frac{2-Q-\Psi_i}{(Q-1)^2} Q.$$

Note that by the first part of Lemma 1 we have $b_i > a_i > 1$ for all i.

We now describe the adversary strategy. Initially, we issue two jobs of some arbitrary weight $x_1 > 0$. Both $\mathcal{A}$ and the adversary execute one job x_1 and we enter the configuration $\langle x_1, x_1\rangle$.

Suppose now that after $i - 1$ iterations, the current configuration is $\langle x_i, x_i\rangle$. The adversary now follows the following steps:

	issue one job $a_i x_i$
(1)	`if` $\mathcal{A}$ executes $a_i x_i$ `then` execute x_i, $a_i x_i$ and halt
	`else` ($\mathcal{A}$ executes x_i)
	at the next time step issue $b_i x_i$
(2)	`if` $\mathcal{A}$ executes $b_i x_i$ `then` execute x_i, $a_i x_i$, $b_i x_i$, and halt
	`else` ($\mathcal{A}$ executes $a_i x_i$)
(3)	at the next time step issue two jobs $x_{i+1} = b_i x_i$
	execute $a_i x_i$, $b_i x_i$, $b_i x_i$

If $\mathcal{A}$ executes first x_i and then $a_i x_i$, then after step (3) it will execute one job $x_{i+1} = b_i x_i$, and the new configuration will be $\langle x_{i+1}, x_{i+1}\rangle$, and the game continues. The adversary strategy is illustrated in Figure 1, where we write $a_i = a$ and $b_i = b$, to simplify the notation. Jobs are represented by line segments, with dark rectangles representing if and when the job is executed by $\mathcal{A}$, while the lightly shaded rectangles represent job executions by the adversary.

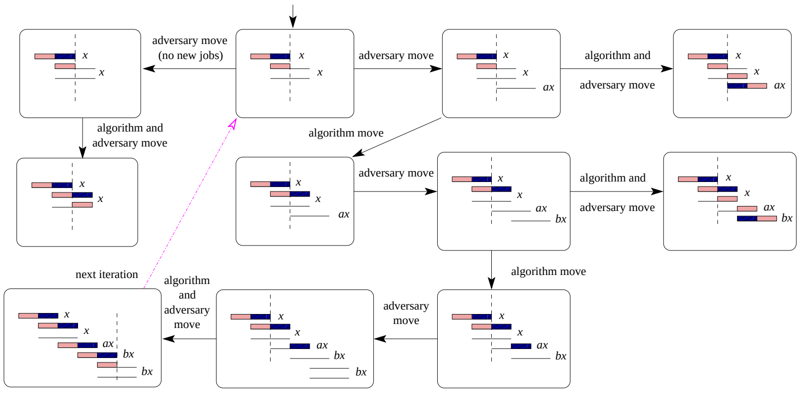

Fig. 1. The adversary strategy.

If the game completes $i-1$ iterations, then define $gain_i$ and adv_i to be the gain of $\mathcal{A}$ and the adversary in these steps (ending in configuration $\langle x_i, x_i \rangle$.)

We claim that for any i, either the adversary wins the game before iteration i, or else in iteration i we have

$$(Q-\epsilon)gain_i - adv_i \le \Psi_i x_i. \tag{1}$$

The proof of (1) is by induction on i. For $i = 1$, $(Q-\epsilon)gain_1 - adv_1 \le (Q-\epsilon)x_1 - x_1 = \Psi_1 x_1$, as claimed. Suppose that (1) holds after iteration $i-1$, that is, when the configuration is $\langle x_i, x_i \rangle$. If $\mathcal{A}$ executes $a_i x_i$ in step (1), then, denoting by ξ the sequence of jobs up to this step, using the inductive assumption, and substituting the formula for a_i, we have

$$\begin{aligned}(Q-\epsilon)gain_{\mathcal{A}}(\xi) - adv(\xi) &= (Q-\epsilon)gain_i - adv_i + (Q-\epsilon)a_i x_i - (x_i + a_i x_i)\\ &\le [\Psi_i - 1 + (Q-\epsilon-1)a_i]x_i \;<\; 0,\end{aligned}$$

contradicting the $(Q-\epsilon)$-competitiveness of $\mathcal{A}$. If $\mathcal{A}$ executes x_i and then $b_i x_i$ in step (2), then, again, denoting by ξ' the sequence of jobs up to this step, using the inductive assumption, and substituting the formulas for a_i, b_i, we have

$$\begin{aligned}(Q-\epsilon)gain_{\mathcal{A}}(\xi') - adv(\xi') &= (Q-\epsilon)(gain_i + x_i + b_i x_i) - (adv_i + x_i + a_i x_i + b_i x_i)\\ &\le [\Psi_i + Q - 1 - \epsilon + (Q-\epsilon-1)b_i - a_i]x_i \;<\; 0,\end{aligned}$$

and this again contradicts the $(Q-\epsilon)$-competitiveness of $\mathcal{A}$.

The only other possibility is that $\mathcal{A}$ will execute first x_i, then $a_i x_i$, and then it has no choice but to execute $b_i x_i$. In the new configuration $\langle x_{i+1}, x_{i+1}\rangle$ we have

$$\begin{aligned}(Q-\epsilon)gain_{i+1} - adv_{i+1} &\leq (Q-\epsilon)(gain_i + x_i + a_i x_i + b_i x_i) - (adv_i + a_i x_i + 2b_i x_i)\\ &\leq [\Psi_i + Q(1+a_i+b_i) - (a_i + 2b_i)]x_i \;=\; b_i x_i \Psi_{i+1} \;=\; x_{i+1}\Psi_{i+1},\end{aligned}$$

completing the proof of (1).

From (1) and from Lemma 1, we obtain that for i large enough we will have $(Q-\epsilon)gain_i - adv_i \leq \Psi_i x_i < (1-Q+\epsilon)x_i$. But this contradicts the $(Q-\epsilon)$-competitiveness of $\mathcal{A}$, since, denoting by ϱ the sequence of all jobs issued up to this time (including the pending jobs x_i), we have

$$\begin{aligned}(Q-\epsilon)gain_{\mathcal{A}}(\varrho) - adv(\varrho) &= (Q-\epsilon)(gain_i + x_i) - (adv_i + x_i)\\ &< (1-Q+\epsilon)x_i + (Q-\epsilon)x_i - x_i \;=\; 0.\end{aligned}$$

5.2 Upper Bound

We now present our Q-competitive algorithm for 2-uniform case. Given that the 2-uniform case seems to be the most elementary case of unit job scheduling (without being trivial), our algorithm (and its analysis) is surprisingly difficult. Recall, however, that, as shown in [2], any algorithm for this case with competitive ratio below $\sqrt{2}$ needs to use some information about the past. Further, when the adversary uses the strategy from Theorem 3, any Q-competitive algorithm needs to behave in an essentially unique way. Our algorithm was designed to match this optimal strategy, and then extended (by interpolation) to other adversarial strategies. Thus we suspect that the complexity of the algorithm is inherent in the problem and cannot be avoided.

We start with some intuitions. Let $\mathcal{A}$ be our online algorithm. Suppose that at time t we have one old pending job z, and two new pending jobs b, c with $b \geq c$. In some cases, the decision which job to execute is easy. If $c \geq z$, $\mathcal{A}$ can ignore z and execute b in the current step. If $z \geq b$, $\mathcal{A}$ can ignore c and execute z in the current step. If $c < z < b$, $\mathcal{A}$ faces a dilemma: it needs to decide whether to execute z or b. Assuming $c = 0$, the choice will be made based on the ratio z/b. If z/b exceeds a certain threshold (possibly dependent on the past), we execute z, otherwise we execute b. Taking those constraints into account, and interpolating for arbitrary values of c, we can handle all cases by introducing a parameter η, $0 \leq \eta \leq 1$, and making the decision according to the following procedure:

Procedure SSP_η**:** If $z \geq \eta b + (1-\eta)c$ schedule z, otherwise schedule b.

To derive an online algorithm, say $\mathcal{A}$, we examine the adversary strategy in the lower bound proof. Consider the limit case, when $i \to \infty$, and let $a_* = \lim_{i\to\infty} a_i = Q/(Q-1)$ and $b_* = \lim_{i\to\infty} b_i = Q/(Q-1)^2$. Assume that in the previous step two jobs z were issued, so that the current configuration is $\langle z, z\rangle$. If the adversary now issues a single job a, then $\mathcal{A}$ needs to do the following: if $a \leq a_* z$, execute z, and if $a \geq a_* z$, then execute a. (The tie for $a = a_* z$ can be broken either way.) Thus in this case we need to apply SSP_α with the threshold $\alpha = 1/a_* = (Q-1)/Q$.

Then, suppose that in the first step $\mathcal{A}$ executed z, so that in the next step a is pending. If the adversary now issues a single job b, then (assuming $a \approx a_*$) $\mathcal{A}$ must to do the following: if $b \leq b_* z$, execute a, and if $b \geq b_* z$, then execute b. Thus in this case we need to apply SSP_β with the threshold $\beta = a_*/b_* = Q-1$.

Suppose that we execute a. In the lower-bound strategy, the adversary would now issue two jobs b in the next step. But what happens if he issues a single job, say c? Calculations show that $\mathcal{A}$, in order to be Q-competitive, needs to use now yet a different parameter η in SSP_η. This parameter is not uniquely determined, but it must be at least $\gamma = (3-2Q)/(2-Q) > Q-1$. Further, it turns out that the same value γ can be used on subsequent single-job requests.

Our algorithm is derived from the above analysis: on a sequence of single-job requests in a row, use SSP_η with parameter α in the first step, then β in the second step, and γ in all subsequent steps. In general, of course, two jobs can be issued at each step (or more, but only two heaviest jobs need to be considered.) We think of an algorithm as a function of several arguments. The values of this function on the boundary are determined from the optimal adversary strategy, as explained above. The remaining values are obtained through interpolation.

We now give a formal description of our algorithm. We define constants

$$\alpha \;=\; \frac{Q-1}{Q} \approx 0.27 \qquad \beta \;=\; Q-1 \approx 0.38 \qquad \gamma \;=\; \frac{3-2Q}{2-Q} \approx 0.39$$

We also use two functions

$$\lambda(\xi) = \min\left\{1, \frac{\xi-\alpha}{\beta-\alpha}\right\}, \qquad \delta(\mu,\xi) = \mu\alpha + (1-\mu)[\beta + (\gamma-\beta)\lambda(\xi)].$$

where $0 \leq \mu \leq 1$ and $\alpha \leq \xi \leq \gamma$. Note that for the parameters μ, ξ within their ranges, we have $0 \leq \lambda(\xi) \leq 1$, $\alpha \leq \delta(\mu,\xi) \leq \gamma$. Function δ also satisfies $\delta(1,\xi) = \alpha$, $\delta(0,\xi) \geq \beta$ for any ξ, and $\delta(0,\alpha) = \beta$, $\delta(0,\beta) = \gamma$.

Algorithm Switch**.** Fix a time step t. Let u, v (where $u \geq v$) be the two jobs released at time $t-1$, and b, c (where $b \geq c$) be the jobs released at time t. Let $z \in \{u, v\}$ be the old pending job. Let also ξ be the parameter of SSP used at time $t-1$ (initially $\xi = \alpha$). Then choose the job to execute as follows:

- If $z = u$ run SSP_η with $\eta = \delta(\frac{v}{u}, \xi)$,
- If $z = v$ run SSP_η with $\eta = \alpha$.

Theorem 4. *Algorithm* Switch *is Q-competitive for the 2-uniform case, where $Q \approx 1.377$ is the largest root of $Q^3 + Q^2 - 4Q + 1 = 0$.*

The proof of the theorem is by tedious case analysis of an appropriate potential function and will appear in the full version of this paper.

6 Conclusions

We established the first upper bound better than 2 on the competitiveness of deterministic scheduling unit jobs to maximize weighted throughput. There is still a wide gap between our upper bound of ≈ 1.939 and the best known lower bound of ϕ. Closing or substantially reducing this gap is a challenging open problem. We point out that our algorithm GenFlag is not memoryless, as it uses one bit of information about the previous step. Whether it is possible to reduce the ratio of 2 with a memoryless algorithm remains an open problem.

Another intriguing open problem is to determine the competitiveness of the case when the jobs are similarly ordered, or when they all have the same span. We gave an algorithm for this version with competitive ratio ≈ 1.838. The lower bound of ϕ applies to similarly ordered jobs. As for the uniform span case, to the best to our knowledge, the best lower bound is ≈ 1.377 given in this paper for the 2-uniform case.

References

1. N. Andelman, Y. Mansour, and A. Zhu. Competitive queueing policies for QoS switches. In *Proc. of the 14th ACM-SIAM SODA*, pages 761–770, 2003.
2. Y. Bartal, F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, R. Lav, J. Sgall, and T. Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. In *Proc. of the 21st STACS*, pages 187–198. Springer, 2004.
3. F. Y. L. Chin and S. P. Y. Fung. Online scheduling for partial job values: does timesharing or randomization help? *Algorithmica*, 37(3):149–164, 2003.
4. 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*, 2001.
5. A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. In *Proc. of the 33rd STOC*, pages 520–529, 2001.
6. A. Kesselman, Y. Mansour, and R. van Stee. Improved competitive guarantees for QoS buffering. In *Proc. of the 11th ESA*, pages 361–372, 2003.