A simpler competitive analysis for scheduling equal-length jobs on one machine with restarts ⋆

Michael H. Goldwasser [a,*] Arundhati Bagchi Misra [b],

[a] *Department of Mathematics and Computer Science, Saint Louis University, St. Louis, MO, USA*
[b] *Department of Mathematics and Statistics, Mississippi State University, Mississippi State, MS, USA*

Abstract

We consider the online problem of scheduling jobs with equal processing times on a single machine. Each job has a release time and a deadline, and the goal is to maximize the number of jobs completed by their deadlines. Chrobak et al. (2007, SICOMP 36:6) introduce a *preempt-restart* model in which progress toward completing a preempted job is lost, yet that job can be restarted from scratch. They provide a $\frac{3}{2}$-competitive deterministic algorithm and show that this is the optimal competitiveness. Their analysis is based on a complex charging scheme among individual jobs and the use of several partial functions and mappings for assigning the charges. In this paper, we provide an alternative proof of the result using a more global potential argument to compare the relative progress of the algorithm versus the optimal schedule over time. This new proof is significantly simpler and more intuitive that the original, and our technique is applicable to related problems.

Key words: analysis of algorithms, on-line algorithms, preemption, scheduling

1. Introduction

Chrobak et al. consider two problems involving the online scheduling of equal-length jobs on a single machine [1]. One of their results involves maximizing the number of jobs that can be completed under a preempt-restart model. They allow a running job to be preempted, yet with all partial work lost. If that job is later executed it must be started from scratch. In this context, they provide a deterministic algorithm that is $\frac{3}{2}$-competitive and a lower bound showing that this is the best possible (deterministic) competitiveness.

As is often the case for scheduling problems, the result is achieved with a relatively straightforward algorithm yet rather complex analysis. In the authors' own words, "we remark that both of our algorithms are natural, easy to state and implement. The competitive analysis is, however, fairly involved, and it relies on some structural lemmas about schedules of equal-length jobs." Indeed, the proof relies upon the declaration of one partial function used to convert the actual schedule produced by the algorithm into a normalized schedule for the sake of analysis. Then a second partial function is formulated for defining a charging scheme whereby individual jobs scheduled by the algorithm are tracked relative to individual jobs scheduled in an optimal schedule.

We demonstrate use of a more global, potential-based argument to prove the same result. Our approach sidesteps the technical challenges of having individual jobs assign charge to individual jobs

⋆ This research is based upon work supported by the National Science Foundation under Grant No. CCR-0417368.
* Corresponding author.
Email addresses: goldwamh@slu.edu (Michael H. Goldwasser), ab659@msstate.edu (Arundhati Bagchi Misra).

and instead measures the cumulative progress of the schedule produced by the algorithm relative to an optimal schedule. We find our analysis more intuitive and technically straightforward than the original. The larger impact of this paper is the development of a proof structure that may be applied to further problems (see, for example, [3,4]).

2. Preliminaries

Each job j is specified by three nonnegative integer parameters, with r_j denoting its release time, p_j its processing time, and d_j its deadline. We subsequently let $x_j = d_j - p_j$ denote a job's *expiration time*, namely the last possible time it can be started in order to meet its deadline. For this paper, we assume all jobs lengths are equal, and thus that $p_j = p$ for some constant p.

We consider an online model in which the scheduler is oblivious to a job's existence until that job's release time; at that point, all parameters are revealed. We consider an algorithm providing what has been termed *immediate notification* [2]. At the moment a job is released, the scheduler must either accept or reject that job. An accepted job need not be scheduled precisely at that moment, but the scheduler must guarantee that it will be successfully completed by its deadline.

3. Algorithm Definition

The algorithm, as presented by Chrobak et al., does not explicitly provide immediate notification. Instead, all released jobs are viewed as pending until the time when they are either completed or expired. However, the authors note that a normalization lemma can be applied in an online fashion to convert their algorithm to such a formulation.

For the remainder of this paper, we rely upon a description of algorithm $\mathcal{A}$ that provides immediate notification, as detailed in Fig. 1. At all times, the algorithm maintains a queue Q of those jobs that have been accepted into the system but not yet completed (including, perhaps, a currently executing job). At the onset of a time step, the algorithm removes any newly completed job from the queue, processes the arrival of all newly released jobs, and, if the queue is nonempty and the machine is currently uncommitted, starts executing the waiting job with *earliest* deadline.

```
1  At time t:
2      if job j has just completed:
3          Q ← Q \ {j}    # no longer pending
4
5      foreach newly arrived job k:
6          if no job is in progress:
7              if FEASIBLE(Q ∪ {k}, t):
8                  accept k
9                  Q ← Q ∪ {k}
10             else:
11                 reject k
12         else:
13             let j denote the running job
14             let s_j > t − p denote its start time
15             if x_k < s_j + p:  # preemption candidate
16                 if FEASIBLE(Q, t + p):
17                     preempt j
18                     accept k
19                     Q ← Q ∪ {k}
20                 else:
21                     reject k
22             else:    # not a preemption candidate
23                 if FEASIBLE((Q \ {j}) ∪ {k}, s_j + p):
24                     accept k
25                     Q ← Q ∪ {k}
26                 else:
27                     reject k
28
29     if no job is in progress:
30         start the earliest-deadline job of Q
```

Fig. 1. The description of algorithm $\mathcal{A}$.

The admission policy for a newly released job is based upon certain feasibility conditions. We use the notation FEASIBLE$(\mathcal{J}, t)$, to denote whether a set $\mathcal{J}$ of jobs can be scheduled starting at time t. The feasibility of a given set of jobs on one machine can be tested using Jackson's rule [5].

The release of a job is handled by one of three distinct cases. If no job is currently being processed (either because the queue is empty or a job has just completed), a newly released job is accepted into the queue so long as it is feasible to schedule that job together with all other accepted jobs (line 7). If some job j is being processed at time t, having been started at $s_j > t - p$, the algorithm behaves as follows. If newly released job k satisfies the condition $x_k < s_j + p$ then it is a *preemption candidate*. So long as the existing queue remains feasible at time $t + p$ (line 16), the currently running job is preempted and k is accepted (and subsequently started, as it will be chosen at line 30); otherwise, k is rejected. If k does

not qualify as a preemption candidate, it is accepted when it is feasible to allow the currently running job to complete with k and the remaining jobs from the queue completed subsequently (line 23).

After evaluating all newly released jobs, the scheduling policy (lines 29–30) is to start running the earliest-deadline job of the queue, presuming the machine is available.

For the sake of discussion, we introduce the following notations. Since the contents of the algorithm's queue changes over time, we let Q_{t-} denote the queue as it exists at line 4 of Fig. 1 for time t, before considering any newly released jobs. We let $Q_{t|k}$ denote the queue as it exists when line 5 is evaluated specifically for job k. Finally, we let Q_{t+} denote the queue as it exists at line 28, after all newly released jobs are considered. Based upon these formulations, we present the following algorithmic property.

Lemma 1 *If job k is rejected by $\mathcal{A}$, there does not exist a time t such that $r_k \leq t \leq x_k$ with* $\textsc{Feasible}(Q_{t+}, t+p)$ *at which $\mathcal{A}$ idles or starts executing a non-preemption-candidate job.*

PROOF. By the algorithm definition, the decision to reject a job is evaluated based on one of three criteria. If job k were rejected when the machine is not yet committed to another job, the condition $\textsc{Feasible}(Q_{r_k|k} \cup \{k\}, r_k)$ must fail. For $t \geq r_k$, jobs of $Q_{r_k|k}$ will have either been completed or remain in Q_{t+}. Therefore $Q_{r_k|k} \cup \{k\}$ is feasible starting at time r_k, by executing $Q_{r_k|k} \setminus Q_{t+}$ over $[r_k, t)$ as in the original schedule, job k at $[t, t+p)$, and Q_{t+} starting at $t+p$. This contradicts the rejection of k.

The second scenario is when k is considered as a preemption candidate, with $\mathcal{A}$ running a job j and $x_k < s_j + p$. However, given that j is executing, there is no way that $\mathcal{A}$ idles or starts a non-preemption-candidate job at time t for $s_j < r_k \leq t \leq x_k < s_j + p$.

Finally, we consider when job j is running and $x_k \geq s_j + p$. In this case, the rejection of k depends on the failure of $\textsc{Feasible}((Q_{r_k|k} \setminus \{j\}) \cup \{k\}, s_j + p)$. Yet we can schedule jobs of $(Q_{r_k|k} \setminus \{j\}) \setminus Q_{t+}$ as done in $\mathcal{A}$, followed by k during $[t, t+p)$, followed by jobs of Q_{t+} from time $t+p$ onward. □

4. Our Analysis

We fix a finite instance $\mathcal{I}$ and an optimal schedule Opt for that instance. Our comparison between $\mathcal{A}$ and Opt is based upon two carefully defined potential functions that measure the respective progress of the developing schedules over time. Before giving a precise definition of those functions, we introduce some additional notation.

To further compare the relative progress of $\mathcal{A}$ and Opt over time, we let $F_t^{\mathcal{A}}$ (resp. F_t^{Opt}) denote the set of jobs completed *strictly* before time $t+p$ by $\mathcal{A}$ (resp. Opt). By this definition, a job of $F_t^{\mathcal{A}}$ is a *fait accompli* at the onset of time t, as it is already progressing toward completion. Note, however, that $F_t^{\mathcal{A}}$ does not include a job that is currently running yet subsequently preempted (because such a job will not be completed prior to $t+p$).

We define $D_t^{\mathcal{A}} = F_t^{\text{Opt}} \cap F_\infty^{\mathcal{A}} \setminus F_t^{\mathcal{A}}$ as the set of "delayed" jobs. These are eventually scheduled by both algorithms and, prior to time t, completed or progressing toward completion by Opt but not by $\mathcal{A}$. Note that $D_t^{\mathcal{A}} \subseteq Q_{t-}$, since these jobs must have arrived strictly before t and been accepted. We define $D_t^{\text{Opt}} = F_t^{\mathcal{A}} \cap F_\infty^{\text{Opt}} \setminus F_t^{\text{Opt}}$ analogously. We now present our potential functions as follows:

$$\Phi_t^{\mathcal{A}} = 3 \cdot |F_t^{\mathcal{A}}| + 1 \cdot |D_t^{\mathcal{A}}|$$

$$\Phi_t^{\text{Opt}} = 2 \cdot |F_t^{\text{Opt}} \cup D_t^{\text{Opt}}|$$

Intuitively, we view these functions as payments to the respective schedules for each job that is a fait accompli as well as for the future promise represented by delayed jobs. In the case of $\mathcal{A}$, we reward 3 units of credit for the former and an advance of 1 unit for jobs qualifying as delayed. In this way, when a delayed job is later started, a balance of 2 additional units are credited to the algorithm. To account for the $\frac{3}{2}$-competitive ratio, we only provide Opt with 2 units of credit per job. However, we give Opt the benefit of the doubt by rewarding delayed jobs with a complete share rather than a partial share.

Our analysis proceeds by identifying times that we term *checkpoints*. We define a checkpoint as a time v at which $\Phi_v^{\mathcal{A}} \geq \Phi_v^{\text{Opt}}$ and at which $\mathcal{A}$ either idles or begins processing a job. By definition, time 0 qualifies as such a checkpoint. In the remainder of our proof, we establish that for any checkpoint u, there exists a checkpoint $v > u$. In this way, we guarantee that $\Phi_\infty^{\mathcal{A}} \geq \Phi_\infty^{\text{Opt}}$ and thus $|F_\infty^{\text{Opt}}| \leq \frac{3}{2}|F_\infty^{\mathcal{A}}|$.

For ease of exposition, we introduce notations $\Phi_{[u,v)}^{\mathcal{A}} = \Phi_v^{\mathcal{A}} - \Phi_u^{\mathcal{A}}$ and $\Phi_{[u,v)}^{\text{Opt}} = \Phi_v^{\text{Opt}} - \Phi_u^{\text{Opt}}$ to measure the relative increases in $\Phi^{\mathcal{A}}$ and Φ^{Opt} over the period $[u, v)$. Given that $\Phi_u^{\mathcal{A}} \geq \Phi_u^{\text{Opt}}$ for checkpoint u, we can demonstrate v as a checkpoint by showing that $\Phi_{[u,v)}^{\mathcal{A}} \geq \Phi_{[u,v)}^{\text{Opt}}$. When analyzing $\Phi_{[u,v)}^{\text{Opt}}$, we rely upon the following partition.

$$(F_v^{\text{OPT}} \cup D_v^{\text{OPT}}) \setminus (F_u^{\text{OPT}} \cup D_u^{\text{OPT}}) =$$
$$(F_v^{\text{OPT}} \setminus F_u^{\text{OPT}}) \setminus F_v^{\mathcal{A}} \ \cup \qquad (\alpha)$$
$$(F_v^{\text{OPT}} \setminus F_u^{\text{OPT}}) \cap (F_v^{\mathcal{A}} \setminus F_u^{\mathcal{A}}) \ \cup \qquad (\beta)$$
$$(F_\infty^{\text{OPT}} \setminus F_v^{\text{OPT}}) \cap (F_v^{\mathcal{A}} \setminus F_u^{\mathcal{A}}) \qquad (\gamma)$$

This identity can be verified algebraically. Intuitively, the set labeled (α) represents jobs that are newly started by OPT but not yet started by $\mathcal{A}$. Set (β) represents jobs that are newly started by both OPT and $\mathcal{A}$ during this period. Set (γ) represents jobs that are started by $\mathcal{A}$ during the period yet held as delayed by OPT. The remainder of our proof proceeds by demonstrating the existence of a checkpoint $v > u$ for three distinct cases.

Case 1: **$\mathcal{A}$ remains idle at time u.**

Lemma 2 *For Case 1, $v = u + 1$ is a checkpoint.*

PROOF. We claim that both $\Phi^{\mathcal{A}}$ and Φ^{OPT} remain unchanged from u to v. Since $\mathcal{A}$ idles over $[u, v)$, no jobs are released at time u, $F_v^{\mathcal{A}} = F_u^{\mathcal{A}}$, and $Q_{u-} = \emptyset$. This implies that $D_v^{\mathcal{A}} = D_u^{\mathcal{A}} = \emptyset$ and $\Phi_{[u,v)}^{\mathcal{A}} = 0$.

We show that $\Phi_{[u,v)}^{\text{OPT}} = 0$ based upon the earlier α, β, γ classification. Since $F_v^{\mathcal{A}} \setminus F_u^{\mathcal{A}} = \emptyset$ for this case, the only jobs that contribute to $\Phi_{[u,v)}^{\text{OPT}}$ are those of $(F_v^{\text{OPT}} \setminus F_u^{\text{OPT}}) \setminus F_v^{\mathcal{A}}$. A job $k \in F_v^{\text{OPT}} \setminus F_u^{\text{OPT}}$ must be started by OPT precisely at time u. Lemma 1 implies that k was accepted by $\mathcal{A}$, which idles at u. Since $Q_{u-} = \emptyset$, it must be that $k \in F_u^{\mathcal{A}} = F_v^{\mathcal{A}}$ and thus $(F_v^{\text{OPT}} \setminus F_u^{\text{OPT}}) \setminus F_v^{\mathcal{A}} = \emptyset$. □

Case 2: **$\mathcal{A}$ starts a job at time u that is then preempted.**

In this case, let j denote the job started at time u, and k denote the preemption candidate for which j was preempted at time r_k. We define $v > r_k$ as the earliest time such that FEASIBLE$(Q_{v-}, v + p)$. This v is well-defined since the queue is eventually empty. Furthermore, some job must be completed at v, since otherwise $v - 1$ suffices.

Lemma 3 *Other than the preemption of j at time r_k, there are no preemptions within a Case 2 region $[u, v)$.*

PROOF. In order for a job k', released at time t, to preempt another it must be that FEASIBLE$(Q_{t|k'}, t + p)$, as described on line 16 of Fig. 1. Because $Q_{t-} \subseteq Q_{t|k'}$, such $r_k < t < v$ violates the presumed minimality of v. □

Lemma 4 *Any job f completed by $\mathcal{A}$ during Case 2 region $[u, v)$ satisfies $x_f < v$.*

PROOF. Let job f be scheduled by $\mathcal{A}$ over interval $[s_f, s_f + p)$, for $u < s_f < s_f + p \leq v$. If $x_f \geq v$, then Q_{s_f-} is feasible at time $s_f + p$, namely by moving f to the period $[v, v + p)$ and scheduling Q_{v-} starting at time $v+p$. This feasibility of Q_{s_f-} at time $s_f + p$ contradicts the definition of v as the earliest such time, and therefore it must be that $x_f < v$. □

Lemma 5 *For Case 2, time v is a checkpoint.*

PROOF. By definition, $\mathcal{A}$ starts job j at time u and preempts it at time r_k. By Lemma 3, there are no further preemptions, and by the definition of v there cannot be any idle time during this region (as an empty queue contradicts the minimality of v). Therefore, $\mathcal{A}$ executes $z \geq 1$ consecutive jobs starting at r_k. Each of those jobs contributes at least 2 toward $\Phi_{[u,v)}^{\mathcal{A}}$, and newly released k nets a third credit as $k \notin D_u^{\mathcal{A}}$. Therefore, $\Phi_{[u,v)}^{\mathcal{A}} \geq 2 \cdot z + 1$.

By Lemma 4, all jobs of $F_v^{\mathcal{A}} \setminus F_u^{\mathcal{A}}$ expire strictly before v. Therefore, the set denoted by γ on page 4 is empty. Combining sets α and β we have that $\Phi_{[u,v)}^{\text{OPT}} = 2 \cdot |(F_v^{\text{OPT}} \setminus F_u^{\text{OPT}}) \setminus F_u^{\mathcal{A}}|$. OPT can *start* at most $z+1$ jobs during the period, as $v-u < (z+1)p$. If there are z or fewer such jobs, then $\Phi_{[u,v)}^{\text{OPT}} \leq 2 \cdot z \leq \Phi_{[u,v)}^{\mathcal{A}}$, thereby validating v as a checkpoint.

We henceforth assume that OPT starts $z + 1$ jobs during the region $[u, v)$, none of which belong to $F_u^{\mathcal{A}}$, thereby achieving a gain of $2 \cdot z + 2$. For this to happen, OPT must start some job $b \neq k$ at time $t < r_k$, as diagrammed in Fig. 2. In this case, we argue that $\mathcal{A}$ must achieve a gain of at least $2 \cdot z + 2$. We already know that $\mathcal{A}$ achieves gain of at least $2 \cdot z + 1$ since $k \notin D_u^{\mathcal{A}}$. If $\mathcal{A}$ completes a second job from outside $D_u^{\mathcal{A}}$, the relative gain is at least $2 \cdot z + 2$ and the claim proven. Otherwise we argue that $b \in D_v^{\mathcal{A}} \setminus D_u^{\mathcal{A}}$, thereby providing the additional unit of gain.

We begin by noting that under these conditions, FEASIBLE$(Q_{r_k|k} \setminus \{j\}, u + 2p)$. When $z = 1$, this is so because $Q_{r_k|k} \subseteq Q_{v-}$, $u + 2p < v + p$, and FEASIBLE$(Q_{v-}, v + p)$ based on the definition of v. Alternatively, if $z > 1$, we can assume that j is

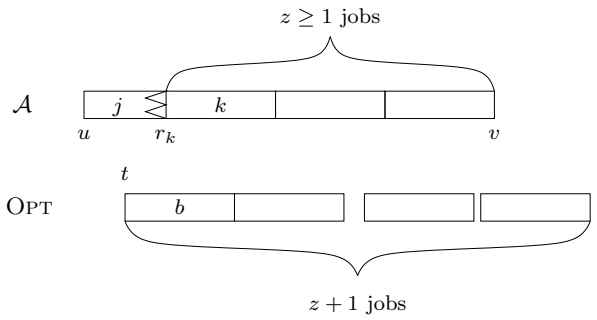

Fig. 2. Case 2 configuration, as considered in proof of Lemma 5.

the job completed immediately after k, as j was the earliest-deadline job from Q_{u+} and all jobs following k in this region are presumed to be in $D_u^{\mathcal{A}} \subseteq Q_{u+}$. In this case, $Q_{r_k|k} \setminus \{j\}$ is scheduled by $\mathcal{A}$ starting at time $r_k + 2p > u + 2p$. This establishes FEASIBLE$(Q_{r_k|k} \setminus \{j\}, u + 2p)$ and as a consequence, FEASIBLE$(Q_{u+}, u + p)$.

We next show that b was accepted by $\mathcal{A}$. If b were released on or before time u, then $r_b \leq u \leq x_b$ and Lemma 1 implies its acceptance by $\mathcal{A}$. If $u < r_b < r_k$ and $x_b < u+p$, then b would have been considered as a preemption candidate. Its rejection would suggest the failure of FEASIBLE$(Q_{r_b|b}, r_b + p)$, contradicting the success of condition FEASIBLE$(Q_{r_k|k}, r_k + p)$ for $Q_{r_k|k} \supseteq Q_{r_b|b}$ and $r_k + p > r_b + p$, as necessitated for the preemption at r_k. Finally, if $u < r_b < r_k$ and $x_b \geq u + p$, its acceptance is guaranteed by condition FEASIBLE$(Q_{r_b|b} \setminus \{j\}, r_b + p)$, noting that set $Q_{r_b|b} \setminus \{j\} \subseteq Q_{r_k|k} \setminus \{j\}$ and $r_b + p < u + 2p$.

By definition, b is neither in $F_u^{\mathcal{A}}$ nor F_u^{OPT}, and thus not in $D_u^{\mathcal{A}}$. Since we have assumed that $k \neq b$ is the only job from outside set $D_u^{\mathcal{A}}$ started by $\mathcal{A}$ during this region, it must be that accepted b remains in Q_{v-} and thus $b \in D_v^{\mathcal{A}} \setminus D_u^{\mathcal{A}}$, providing 1 new credit toward $\Phi_{[u,v)}^{\mathcal{A}} \geq 2 \cdot z + 2 = \Phi_{[u,v)}^{\text{OPT}}$. □

Case 3: **$\mathcal{A}$ starts a job at time u which then runs to completion.**

Let j denote the job started by $\mathcal{A}$ at time u. If OPT starts a job b at a time t such that $u \leq t < u + p$ we define $\epsilon < p$ such that $t = u + \epsilon$; otherwise we let $\epsilon = 0$. Now we let $v \geq u + p$ denote the first such time that FEASIBLE$(Q_{v-}, u + (z + 1) \cdot p + \epsilon)$, where z denotes the number of jobs completed by $\mathcal{A}$ during $[u, v)$. As with Case 2, v is well-defined and aligned with the completion of a job.

Lemma 6 *During a Case 3 region $[u, v)$, the cumulative time spent on preempted portions of jobs is at most ϵ.*

PROOF. For contradiction, let a be the preempted job that causes those cumulative efforts to surpass the bound of ϵ. Let $z' \geq 1$ denote the number of jobs completed during this region prior to the time $s_a \geq u + p$ when a is started. For a to be preempted at a time $r_{k'} > u + z' \cdot p + \epsilon$, it must be that FEASIBLE$(Q_{r_{k'}|k'}, r_{k'} + p)$. Given that $Q_{r_{k'}|k'} \supseteq Q_{s_a-}$ and $r_{k'} + p > u + (z' + 1) \cdot p + \epsilon$, this implies FEASIBLE$(Q_{s_a-}, u + (z' + 1) \cdot p + \epsilon)$. Such $u + p \leq s_a < v$ contradicts the defined minimality of v. □

Lemma 7 *For any job, $a \neq j$ completed by $\mathcal{A}$ during the Case 3 region, $x_a < u + z \cdot p + \epsilon$.*

PROOF. Assume for contradiction that job $a \neq j$ with $x_a \geq u + z \cdot p + \epsilon$ is the latest such job completed during the region. We let s_a denote the starting time of a. Let $z' \leq z - 1$ denote the number of jobs completed prior to job a. We claim that FEASIBLE$(Q_{s_a-}, u + z \cdot p + \epsilon)$, contradicting the minimality of v, as $u + z \cdot p + \epsilon \geq u + (z' + 1) \cdot p + \epsilon$.

We can schedule a at time $u+z\cdot p+\epsilon$. By definition of v, set Q_{v-} is feasible at time $u + (z + 1) \cdot p + \epsilon$. Furthermore, we see that Q_{v-} includes all jobs of $Q_{s_a-} \setminus \{a\}$; by the definition of a, all subsequently completed jobs have expiration strictly before x_a yet none of those could be in Q_{s_a-} since a had the earliest deadline at that time. □

Lemma 8 *For Case 3, time v is a checkpoint.*

PROOF. We claim that there are at most $z + 1$ jobs contributing to $\Phi_{[u,v)}^{\text{OPT}}$. By Lemma 6, algorithm $\mathcal{A}$ spends at most ϵ time on preempted efforts, and so $v \leq u+z\cdot p+\epsilon$. Furthermore, OPT does not start any job during the interval $[u, u+\epsilon)$ by the definition of ϵ. Therefore OPT can start at most z jobs during the interval $[u+\epsilon, u+z\cdot p+\epsilon)$. Jobs started by OPT on or after $u + z \cdot p + \epsilon \geq v$ can only contribute to $\Phi_{[u,v)}^{\text{OPT}}$ if they belong to $F_v^{\mathcal{A}} \setminus F_u^{\mathcal{A}}$. By Lemma 7 all jobs of $F_v^{\mathcal{A}} \setminus F_u^{\mathcal{A}}$ other than j expire strictly before time $u + z \cdot p + \epsilon$, so we conclude that there are at most $z + 1$ jobs contributing toward $\Phi_{[u,v)}^{\text{OPT}}$.

We see that $\Phi_{[u,v)}^{\mathcal{A}} \geq 2 \cdot z$, as each of the z jobs scheduled by $\mathcal{A}$ results in at least 2 units of credit. If

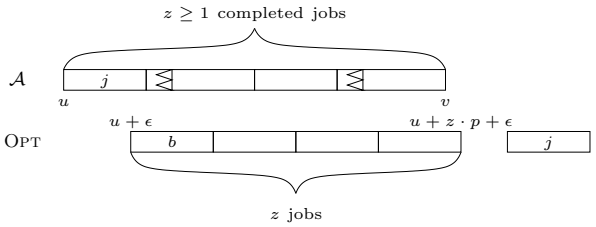

Fig. 3. Case 3 configuration, as considered in proof of Lemma 8.

$\Phi^{\text{OPT}}_{[u,v)} \leq 2 \cdot z$ the claim is proven. So for the remainder of this proof, we concern ourself with the scenario, diagrammed in Fig. 3, in which $\Phi^{\text{OPT}}_{[u,v)} = 2 \cdot z + 2$.

By Lemma 7, all jobs scheduled by $\mathcal{A}$, other than j, expire strictly before $u + z \cdot p + \epsilon$. Yet $x_j \geq u + z \cdot p + \epsilon$, as evidenced by OPT's placement. Since j was chosen at time u as the earliest deadline job of Q_{u+}, we infer that all other jobs scheduled by $\mathcal{A}$ must have arrived strictly after time u. None of those jobs could belong to $D^{\mathcal{A}}_u$ (nor could j, given that it is presumed to be delayed in OPT). Therefore, $\Phi^{\mathcal{A}}_{[u,v)} = 3 \cdot z$. If $z \geq 2$, then $3 \cdot z \geq 2 \cdot z + 2$ and the proof is completed.

We need only consider the special case of $z = 1$. In this case $\Phi^{\text{OPT}}_{[u,v)} = 4$, and $\Phi^{\mathcal{A}}_{[u,v)} \geq 3$ due to job $j \in F^{\mathcal{A}}_v \setminus (F^{\mathcal{A}}_u \cup D^{\mathcal{A}}_u)$. We assume OPT is credited for $b \notin D^{\mathcal{A}}_u$, and therefore that b has not previously been started by $\mathcal{A}$. We argue that $\mathcal{A}$ could not have rejected b, and thus $b \in Q_{v-}$ contributes 1 additional unit toward $\Phi^{\mathcal{A}}_{[u,v)}$ as a member of $D^{\mathcal{A}}_v \setminus D^{\mathcal{A}}_u$. By the definition of v, $\text{FEASIBLE}(Q_{v-}, v + p + \epsilon)$. Based upon the placement of j by $\mathcal{A}$ and OPT, $r_j \leq u \leq v + \epsilon \leq x_j$. Therefore, $\text{FEASIBLE}(Q_{v-} \cup \{j\}, v + \epsilon)$. We demonstrate the acceptance of b by analyzing three sub-cases.

- When $r_b \leq u$ we apply Lemma 1, noting that $Q_{u+} \subseteq Q_{v-} \cup \{j\}$ and thus $\text{FEASIBLE}(Q_{u+}, u + p)$.
- When $r_b > u$ yet $x_b < u + p$, job b is a preemption candidate. Since $Q_{r_b|b} \subseteq Q_{v-} \cup \{j\}$ and $r_b + p \leq v + \epsilon$, we have $\text{FEASIBLE}(Q_{r_b|b}, r_b + p)$, implying that b would have been accepted.
- If $r_b > u$ and $x_b \geq u + p$, the acceptance of b is based on $\text{FEASIBLE}((Q_{r_b|b} \setminus \{j\}) \cup \{b\}, v)$, noting that b could be executed during $[v, v + p)$ and $Q_{r_b|b} \setminus \{j\} \subseteq Q_{v-}$ starting at $v + p$.

This completes the proof. □

Theorem 9 *Algorithm $\mathcal{A}$ is $\frac{3}{2}$-competitive.*

PROOF. We claim by induction that there exists a checkpoint v at which both schedules are complete. Time 0 is trivially a checkpoint, and the case analysis of Lemmas 2, 5, and 8 assures that we can continue to find later checkpoints. As no jobs are delayed for such v, the condition $\Phi^{\mathcal{A}}_v \geq \Phi^{\text{OPT}}_v$ can be simplified to $3 \cdot |F^{\mathcal{A}}_v| \geq 2 \cdot |F^{\text{OPT}}_v|$, and thus $\frac{|\text{OPT}|}{|\mathcal{A}|} \leq \frac{3}{2}$. □

5. Conclusions

We have re-derived a $\frac{3}{2}$-competitive bound for scheduling equal-length jobs on one machine with a preempt-restart model. For this problem, our analysis via global potential functions is significantly different from the charging scheme used originally by Chrobak et al. [1]. We feel that our analysis is technically simpler and more intuitive.

More significantly, the style of analysis we use can be applied to other related problems. For example, such a potential function argument was recently used to prove a new result for maximizing the number of equal-length jobs when scheduling two machines, non-preemptively [3,4].

References

[1] M. Chrobak, W. Jawor, J. Sgall, T. Tichý, Online scheduling of equal-length jobs: Randomization and restarts help, SIAM J. Comput. 36 (6) (2007) 1709–1728.

[2] M. H. Goldwasser, B. Kerbikov, Admission control with immediate notification, J. Scheduling 6 (3) (2003) 269–285.

[3] M. H. Goldwasser, M. Pedigo, Online, non-preemptive scheduling of equal-length jobs on two identical machines, in: L. Arge, R. Freivalds (eds.), Proc. 10th Scandinavian Workshop on Algorithm Theory (SWAT), vol. 4059 of Lecture Notes in Computer Science, Springer-Verlag, Riga, Latvia, 2006.

[4] M. H. Goldwasser, M. Pedigo, Online, non-preemptive scheduling of equal-length jobs on two identical machines, ACM Trans. on Algorithms. To appear.

[5] J. R. Jackson, Scheduling a production line to minimize maximum tardiness, Research Report 43, Management Science Research Project, University of California, Los Angeles (Jan. 1955).