SCHEDULING ON A SINGLE MACHINE TO MINIMIZE TOTAL FLOW TIME WITH JOB REJECTIONS

David P. Bunde*
University of Illinois at Urbana-Champaign
bunde@uiuc.edu

Abstract We consider the problem of minimizing flow time on a single machine supporting preemption that can reject jobs at a cost. Even if all jobs have the same rejection cost, we show that no online algorithm can have competitive ratio better than $(2+\sqrt{2})/2 \approx 1.707$ in general or $e/(e-1) \approx 1.582$ if all jobs are known to have the same processing time. We also give an optimal offline algorithm for unit-length jobs with arbitrary rejection costs. This leads to a pair of 2-competitive online algorithms for unit-length jobs, one when all rejection costs are equal and one when they are arbitrary. Finally, we show that the offline problem is NP-hard even when each job's rejection cost is proportional to its processing time.

Keywords: Complexity of Scheduling Problems, Theoretical Scheduling

1 Introduction

In classical scheduling problems, the input is a stream of jobs, all of which must be completed. This type of problem is just one aspect of planning for industrial production because it ignores the process of deciding what jobs to complete. One view of industrial planning divides it into four levels, motivated by semiconductor manufacturing [9]. At the top level, management decides which product lines to promote and how much production to devote to each. The second level decides on the desired production for specific products and the third converts this into requests for components that must be completed by specific times. At the bottom level is a classical scheduling problem involving the assignment of tasks to specific machinery. Classical scheduling ignores the decisions at the top levels since they do not fit neatly into a mathematical framework, but these decisions cannot be made without some assurance that the resulting goals are feasible, requiring input from the scheduler. Although interaction between levels may be limited in many cases, it would be advantageous to combine the job selection and scheduling decisions.

In this paper, we consider one such combination. Our system receives requests that potentially exceed its capacity and focuses on a subset of them, satisfying this subset while rejecting the rest. More formally, we attempt to schedule n jobs on a single machine. Each job J_i completes after running exclusively on the machine for its *processing time* p_i, incurs a *rejection cost* c_i if it is rejected, and cannot be run or rejected before its *release time* r_i. The time at which algorithm A completes or rejects job J_i is its *completion time*, denoted C_i^A. The *flow time* F_i^A of job J_i is $C_i^A - r_i$, the time between J_i's arrival and its completion or rejection. The *cost* of job J_i, denoted $cost_i^A$, is its flow time (F_i^A) if algorithm A completes job J_i or its flow time plus its rejection cost ($F_i^A + c_i$) if job J_i is rejected. The objective is to minimize $\sum_{i=1}^{n} cost_i^A$, the total cost. This paper considers machines supporting *preemption*, meaning jobs can be interrupted and resumed later without penalty. (This model is called *preempt-resume*.) We primarily consider the *online* problem, where the algorithm

*Partially supported by NSF grant CCR 0093348.

has no knowledge of a job until its release time, but we also consider the *offline* problem, where all aspects of the input are known initially.

Although this problem differs from the factory setting described above, we believe it is an interesting way to add an element of planning to the scheduling decision. The total flow time metric accurately captures the quality of service for an interactive system. Bansal et al. [2] discuss minimizing total flow time with rejections in the context of managing a "to do" list. Throughout the day, you receive tasks from coworkers. Each comes with two (potentially unknown) parameters, the amount of work it will take you and the time it will take someone else. These correspond to the task's processing time and its rejection cost, respectively. The time a coworker waits for a requested task is either your time to complete it, or the time you wait before delegating it to someone else plus their time to complete it. Minimizing the sum of coworker waiting times is exactly our metric.

Related work. Scheduling to minimize flow time without rejections has been extensively studied; see the survey by Pruhs et al. [10]. In the uniprocessor preemptive setting, the algorithm Shortest Remaining Processing Time (SRPT), which always runs the job with the least processing time remaining, is known to be optimal [12, 15]. In other settings, however, optimal solutions cannot always be found. When this occurs, one goal is to give algorithms with performance guarantees. We say that an online algorithm is *c-competitive* or an offline algorithm is a *c-approximation* if the cost of its solution is never more than c times the cost of the optimal solution.

Minimizing flow time with rejections has previously been considered by Bansal et al. [2], whose work is included in Bansal's dissertation [1]. When the rejection costs are arbitrary, they show that no randomized algorithm is better than $\sqrt[4]{n}$ or $\sqrt{C}$-competitive, where C is the ratio between the largest and smallest rejection costs. When all jobs have the same rejection cost, they give a 2-competitive algorithm and a lower bound of 1.5 that holds even when all jobs have processing time 1. Using ideas of Bansal and Dhandhere [3], they generalize the algorithm to the weighted problem by rounding weights to powers of 2. The result is $O(\log^2 W)$-competitive if all jobs in a weight class have the same rejection cost, where W is the ratio between the largest and smallest weights. They also give an algorithm for general rejection costs that is $O(\frac{1}{\epsilon}(\log W + \log C)^2)$-competitive if it runs on a processor that is $(1+\epsilon)$ times faster than the optimal solution's processor. Finally, they mention a $(1+\epsilon)$-approximation with running time $n^{O(\log n/\epsilon^2)}$ for the offline unweighted problem when all jobs have the same rejection cost.

Engels et al. [6] consider offline algorithms to minimize a related metric, weighted completion time ($\sum_i w_i C_i^A$ + cost of rejections). They show this problem is NP-complete even when all jobs arrive at time 0, have weight equal to their processing time, and have the same rejection cost. (This implies that minimizing weighted flow time is NP-hard under the same conditions.) They also give optimal algorithms for some special cases when all jobs arrive at time 0 on a single machine and approximation algorithms for scheduling multiple unrelated machines.

Scheduling with rejections was introduced by Bartal et al. [4], who tried to minimize makespan ($\max_i C_i^A$ + cost of rejections) on a nonpreemptive multiprocessor. Seiden [13] gives better bounds by allowing preemption and Hoogeveen [8] considers the offline problem with preemption. Sengupta [14] considers scheduling with rejections to minimize maximum lateness and tardiness.

Our results. We give both algorithms and lower bounds for minimizing flow time with rejections. Many of our results require the assumption that all processing times are the same and each arrival time is a multiple of this value. We call this special case *unit-length jobs* and assume the processing times are 1 and job arrival times are integers. This corresponds to an application with events occurring on clock ticks and/or extremely short jobs. Focusing on unit-length jobs allows us to assume that any job started is completed before another arrives. Our results are the following:

- When all jobs have the same rejection cost, we show that no algorithm has a constant competitive ratio below $e/(e-1) \approx 1.582$ if all jobs have unit length or $(2+\sqrt{2})/2 \approx 1.707$

if distinct processing times are allowed. The previous best lower bound in both cases was 1.5 [2].

- We give an optimal offline algorithm and a 2-competitive online algorithm for the special case of unit-length jobs with arbitrary rejection costs. Nothing was previously known in this case.
- We introduce a family of online algorithms and show that one member is 2-competitive for unit-length jobs, each having the same rejection cost. This result is not surprising since the algorithm above achieves the same competitive ratio with arbitrary rejection costs and the algorithm of Bansal et al. [2] achieves it with arbitrary processing times. We include it, however, because both the algorithm and proof technique are different.
- We show that the problem is NP-complete even when the rejection cost of a job is proportional to its processing time. This is the first offline hardness result for the unweighted problem, answering an open question posed by Bansal [1].

We also have, but defer to a later version of this paper, an optimal offline algorithm for the special case when all jobs have the same (non-unit) processing time.

The rest of this paper is organized as follows. Section 2 contains our lower bounds on the competitive ratio of online algorithms. Section 3 gives algorithms for unit-length jobs. Section 4 contains the NP-completeness proof. Finally, Section 5 concludes with a discussion of open problems.

2 Lower bounds

Now we give lower bounds on the competitive ratio when all jobs have the same rejection cost, which we denote with c. In our proofs, we use OPT to refer to the optimal algorithm and ALG to refer to a hypothetical algorithm whose competitive ratio is better than our bounds.

2.1 When all jobs have unit length

Our first lower bound applies to unit-length jobs. It is a generalization of one presented by Bansal et al. [2]. The instance in their bound has two parts. The first is a stream of jobs, one released at each unit time. The second is a single "extra" job, released at time 0. The stream continues until the algorithm rejects a job, which we may assume to be the extra one. Their lower bound of 3/2 comes from the tradeoff between rejecting the extra job quickly, in which case OPT finishes all jobs quickly and there is little to amortize the rejection cost against, and rejecting the extra job later, in which case OPT rejects it immediately and the algorithm's flow time is too large.

Our lower bound instance consists of a stream of jobs arriving one per time unit and k extra jobs released at time 0. We will show that the algorithm cannot achieve a competitive ratio better than $e/(e-1)$ because it must wait until a time $\tau_{k,i}c$ to make the i^{th} rejection, but that delaying the rejections to these times yields the same competitive ratio. We define $\tau_{k,i}$ recursively as follows:

$$\begin{aligned} \tau_{k,0} &= 0 \\ \tau_{k,i} &= \frac{i + \sum_{j=1}^{i-1} \tau_{k,j} - (i-1)\tau_{k,i}}{(k+1)/(e-1)}. \end{aligned} \tag{1}$$

Subtracting $\tau_{k,i-1}$ from $\tau_{k,i}$ and solving for the difference gives the following alternate definition:

$$\tau_{k,i} = \sum_{j=1}^{i} \frac{1}{(k+1)/(e-1) + (j-1)} \tag{2}$$

In this paper, we use the following observations about $\tau_{k,k}$:

LEMMA 1 *For all values of k, $\tau_{k,k} < 1$.*

LEMMA 2 $\lim_{k\to\infty} \tau_{k,k} = 1$

These lemmas have technical and unenlightening proofs, which appear in the Appendix. Instead, we proceed with the lower bound:

THEOREM 3 *No deterministic algorithm for flow time with rejections is $(e/(e-1)-\epsilon)$-competitive for any constant $\epsilon > 0$ even when all jobs have unit length and the same rejection cost.*

Proof: Suppose to the contrary that ALG is $(e/(e-1)-\epsilon)$-competitive. Choose k large enough so $\tau_{k,k} \geq 1-\epsilon/2$, which is possible by Lemma 2. We consider the instance described above consisting of a stream of jobs, one released at each unit time, and k extra jobs released at time 0. All jobs have unit length and rejection cost $c = 6k^2/\epsilon$.

Let time t_i be when ALG makes its i^{th} rejection, assuming the stream continues until time t_i. We consider two cases. The first is $t_i < \tau_{k,i}c$ for some smallest value i. In this case, the last job of the stream arrives at time t_i. ALG has cost at least $ic + \sum_{j=1}^{i} t_j + (k-i+1)t_i + \binom{k+1-i}{2} > ic + \sum_{j=1}^{i} t_j + (k-i+1)t_i$. OPT has cost at most $(k+1)t_i + \binom{k+1}{2}$. If $i = 1$, the competitive ratio is at least $ec/((e-1)c+k^2) > e/(e-1)-\epsilon$ and we are done. If $i > 1$, the cost of OPT is at most $(k+1)t_i + \binom{k+1}{2} \leq (k+1)t_i + k^2 \leq (k+1)t_i(1+\epsilon/(3(e-1))) \leq (k+1)t_i(1+\epsilon(e-1)/e)$. Algebraic manipulation of the ratio between these bounds leads to a contradiction:

$$\begin{aligned}
\frac{ic + \sum_{j=1}^{i} t_j + (k-i+1)t_i}{(k+1)t_i(1+\epsilon(e-1)/e)} &= \frac{1}{1+\epsilon(e-1)/e} + \frac{ic + \sum_{j=1}^{i-1} t_j - (i-1)t_i}{(k+1)t_i(1+\epsilon(e-1)/e)} \\
&\geq \frac{1}{1+\epsilon(e-1)/e} + \frac{ic + \sum_{j=1}^{i-1} \tau_{k,j}c - (i-1)\tau_{k,i}c}{(k+1)\tau_{k,i}c(1+\epsilon(e-1)/e)} \\
&= \frac{e}{(e-1)(1+\epsilon(e-1)/e)} \\
&> \frac{e}{e-1} - \epsilon.
\end{aligned}$$

In the second case, $t_i \geq \tau_{k,i}c$ for each i. In this case, the last job of the stream arrives at time t_k. The cost of ALG is $kc + \sum_{i=1}^{k} t_i + t_k + 1 \geq kc + \sum_{i=1}^{k-1} \tau_{k,i}c + 2t_k + 1$. OPT can immediately reject all k extra jobs, so its cost is at most $kc + t_k + 1$. Again, manipulation of the ratio between these bounds leads to a contradiction:

$$\begin{aligned}
\frac{kc + \sum_{i=1}^{k} \tau_{k,i} + t_k + 1}{kc + t_k + 1} &\geq \frac{kc + \sum_{i=1}^{k} \tau_{k,i}c + \tau_{k,k}c + 1}{kc + \tau_{k,k}c + 1} = 1 + \frac{\sum_{i=1}^{k} \tau_{k,i}}{kc + \tau_{k,k}c + 1} \\
&> 1 + \frac{\sum_{i=1}^{k} \tau_{k,i}}{k + 1 + 1/c} > 1 + \frac{\sum_{i=1}^{k} \tau_{k,i}}{(k+1)(1+\epsilon/6)} \\
&\geq 1 + \frac{1}{(e-1)(1+\epsilon/6)} - \frac{k\epsilon}{2(k+1)(1+\epsilon/6)} \\
&\geq 1 + \frac{1-\epsilon/3}{e-1} - \epsilon/2 \\
&> \frac{e}{e-1} - \epsilon.
\end{aligned}$$

□

2.2 When jobs have arbitrary processing times

For jobs with arbitrary processing times, we prove the following stronger lower bound:

THEOREM 4 *When jobs have uniform rejection cost and arbitrary processing time, no deterministic online algorithm has a constant competitive ratio better than* $(2+\sqrt{2})/2 \approx 1.707$.

Proof: Let $\gamma = (2+\sqrt{2})/2$. Suppose to the contrary that a deterministic algorithm ALG is $(\gamma-\epsilon)$-competitive. Without loss of generality, we assume $1/\epsilon$ is an integer. Let $c = 2\gamma/\epsilon$.

Using these values, we give an input instance on which ALG has competitive ratio worse than $\gamma-\epsilon$. At time 0, release a job with processing time $c/\gamma = 2/\epsilon$. Starting at time $c/\gamma - 1$, release a job with processing time 1 every unit time until ALG rejects a job at some time t. If $t < c/\gamma - 1$, the instance has only one job and the competitive ratio is at least $c/(c/\gamma) = \gamma > \gamma - \epsilon$. Otherwise, the competitive ratio is

$$\frac{2t - (c/\gamma - 1) + c + 1}{\min\{t+1, c\} + t + 1 - (c/\gamma - 1)}.$$

This is minimized when $t = c - 1$, so this ratio is at least

$$\frac{3c - c/\gamma}{2c - c/\gamma + 1}.$$

Since the ratio is at most 2, this is greater than

$$\frac{3c - c/\gamma - 2}{2c - c/\gamma} > \gamma - 2/c > \gamma - \epsilon.$$

□

Intuitively, the bound is improved by the flow time incurred before ALG rejects the long job.

3 Algorithms for unit-length jobs

Now we give algorithms for the special case of unit-length jobs. We describe these algorithms in terms of the *active* jobs at a particular time, those that have arrived but have not yet been completed or rejected.

3.1 Optimal offline algorithm

We begin by observing which jobs an offline optimal algorithm rejects. We call $r_i + c_i$ the *rank* of job J_i and denote it with $rank(J_i)$.

LEMMA 5 *If all jobs have unit processing time and OPT rejects job* J_i*, then no job with rank lower than* $rank(J_i)$ *is active any time during the interval* $[r_i, r_i + c_i)$.

Proof: Suppose to the contrary that OPT rejects some job J_i but completes a job J_j during $[r_i, r_i + c_i)$, with $r_j + c_j < r_i + c_i$. Consider schedule OPT' that is identical to OPT except that OPT' rejects J_j at time r_j and runs J_i in the time freed up, i.e. starting at time $C_j^{OPT} - 1$ and ending at C_j^{OPT}. Jobs J_i and J_j contribute $c_i + C_j^{OPT} - r_j$ to OPT and $C_i^{OPT'} - r_i + c_j$ to OPT'. By construction, $C_i^{OPT'} = C_j^{OPT}$ and the cost of every job other than J_i and J_j is the same. Thus, $cost(OPT') - cost(OPT) = -r_i + c_j - (c_i - r_j) = r_j + c_j - (r_i + c_i) < 0$. Therefore, schedule OPT' is strictly better than OPT, a contradiction. □

Implicit in the proof of this lemma is that a rejected job can be swapped for a job of equal rank without affecting the total cost. Let a *normalized optimal solution* be an optimal solution that rejects the fewest number of jobs and always runs the active job with highest rank, breaking ties with the job number.

Lemma 5 leads us to the algorithm Offline Highest Rank First (*HRF–OFF*). At all times, *HRF–OFF* runs the active job with highest rank, again using job numbers to break ties. It rejects any job whose flow time equals its rejection cost, i.e., any job not completed by time equal to its rank. Each of *HRF–OFF*'s rejections occurs at the rejected job's arrival time.

THEOREM 6 *Algorithm* HRF–OFF *is optimal for unit-length jobs with arbitrary rejection costs.*

Proof: Let OPT be a normalized optimal solution. By construction, OPT and *HRF–OFF* can only differ if they reject different jobs. Consider the first time this occurs. Since both algorithms always reject jobs when they arrive, this time is the arrival time r_i of some job J_i.

Suppose *HRF–OFF* rejects job J_i and OPT does not. This is the first difference between the schedules so OPT and *HRF–OFF* have the same active jobs just prior to time r_i. Also, Lemma 5 implies that OPT does not reject jobs ranked above J_i arriving between times r_i and C_i^{OPT}. Since both OPT and *HRF–OFF* run jobs in order of decreasing rank, they have the same higher-ranked active jobs throughout this interval. *HRF–OFF* is busy with higher-ranked jobs until time $r_i + c_i$ because it rejects job J_i. Thus, the flow time of job J_i in OPT is greater than c_i, a contradiction.

Now suppose OPT rejects job J_i and *HRF–OFF* does not. Let $t = C_i^{HRF-OFF} - 1$ be the time *HRF–OFF* starts job J_i. At time t, *HRF–OFF* has completed all jobs ranked above J_i that have arrived because it runs jobs in order of decreasing rank. (None are rejected since a job is rejected exactly when it cannot be completed by time equal to its rank and $t < rank(J_i) \leq$ "higher" rank.) OPT also runs jobs in order of decreasing rank so it has also completed these jobs. By Lemma 5, OPT has no lower-ranked active jobs at time t since $r_i \leq t < rank(J_i) = r_i + c_i$. Thus, it has no active jobs at time t and running job J_i then instead of rejecting it is a valid schedule. This schedule has fewer rejections than OPT and no worse cost, a contradiction. □

3.2 2-competitive online algorithm

HRF–OFF makes most of its decisions online, but is an offline algorithm since running it online requires the ability to reject jobs retroactively. We call the algorithm that rejects jobs when their flow time equals their rejection cost Online Highest Rank First (*HRF–ON*) since it can run online.

By Theorem 6, *HRF–ON* is optimal if each rejection is made retroactively at the job's release time. Since each rejected job J_i accumulates flow time c_i before rejection, we get the following:

THEOREM 7 HRF–ON *is 2-competitive for unit-length jobs having arbitrary rejection costs.*

In fact, *HRF–ON* is not $(2-\epsilon)$-competitive for any constant $\epsilon > 0$ even if we assume it rejects jobs once it is clear their flow time will exceed their rejection cost. Consider the following instance where all jobs have rejection cost c: $\sqrt{c}$ jobs (the *main group*) arrive at time 0 and another job arrives at each time i for $i = 0, \ldots, c-1$ (the *stream*). The algorithm keeps all the main group jobs until time $c - \sqrt{c}$, at which point it rejects them one at a time, one per unit time. Thus, these jobs incur cost $\sqrt{c}(c-\sqrt{c}) + \binom{\sqrt{c}}{2} + c\sqrt{c} = 2c^{3/2} - c/2 + \sqrt{c}/2$. Jobs in the stream run as they arrive, for combined flow time and cost c. Thus, the total cost is $2c^{3/2} + c/2 + \sqrt{c}/2$. The optimal algorithm rejects the main group immediately and runs the stream as it arrives, for total cost $c^{3/2} + c$. Thus, the competitive ratio approaches 2 as c increases.

3.3 Online algorithm when all jobs have rejection cost c

Let the *psuedoschedule* of a schedule be the modification created by keeping each rejected job J_i active for c_i time units. Intuitively, these jobs wait until being instantly finished or rejected for free. Note that the psuedoschedule's total flow is the cost of the schedule from which it is made. When this schedule is an optimal schedule, we call the result an *optimal pseudoschedule*. The optimal psuedoschedule can be generated by an online simulation of *HRF–OFF* since *HRF–OFF* is optimal and decides whether to reject each job J_i within time c_i of its release.

Using this simulation, we define a family of algorithms called *ratio rejecting*. A member of this family is denoted $RR(\rho)$ for a constant $\rho > 1$, though we omit ρ when discussing a general property of the family or when its value is clear from context. Let $cost(EOPT, t)$ and $cost(RR(\rho), t)$ be the costs incurred up to time t by the simulation and $RR(\rho)$, respectively. $RR(\rho)$ always runs the most recently-arrived job. It rejects the active job with earliest arrival time whenever

$$\frac{cost(RR(\rho), t) + c}{cost(EOPT, t)} \leq \rho. \tag{3}$$

Since $cost(EOPT, t)$ is a lower bound on the optimal cost, this condition ensures that the competitive ratio is at most ρ when $RR(\rho)$ makes a rejection. This observation quickly leads to the following lemma:

LEMMA 8 *$RR(\rho)$ is ρ-competitive if it completes or rejects each job within time c of its release.*

Proof: Let OPT be an optimal psuedoschedule. We claim that each job is active in OPT at least as long as in RR. For jobs that OPT rejects, this follows immediately from our assumption on job completion/rejection times since these jobs are active in OPT for time c after their release. Suppose to the contrary that there exists a job that OPT runs before RR. Let J_i be the job with earliest completion time in RR having this property. Let J_j be the job RR completed immediately before J_i. Because RR runs the job with latest arrival time, $r_j \geq r_i$. OPT does not reject job J_j since it does not reject job J_i. Also, because it uses the same criteria to select which job to run, OPT must have completed job J_j before running job J_i. Thus, $C_j^{OPT} < C_i^{OPT} \leq C_j^{RR}$, contradicting our assumption that job J_i was the first job that OPT runs before RR.

Now the lemma follows by considering how the ratio $cost(RR, t)/cost(EOPT, t)$ changes over time. Since each job is active in OPT at least as long as in RR, $cost(EOPT, t)$ increases at least as fast as $cost(RR, t)$ except when RR rejects a job. Thus, the ratio declines toward 1 except when RR rejects a job. By definition, however, RR only rejects a job when doing so does not raise the ratio above ρ. Therefore, the ratio lies between 1 and ρ at all times. Since this ratio at the end of the input is exactly the competitive ratio, the competitive ratio of $RR(\rho)$ is at most ρ. □

Using this, we can show that $RR(2)$ is 2-competitive:

THEOREM 9 *$RR(2)$ is 2-competitive for unit-length jobs all having rejection cost c.*

Proof: Call all jobs that $RR(\rho)$ does not run within time c of their release *extra* jobs. (Note that this includes all jobs RR rejects.) We show that all extra jobs are rejected within time c of their release; the result then follows from Lemma 8. First we consider the simple case when all extra jobs arrive at a single time. Without loss of generality, assume this time is 0. Let k be the number of extra jobs and let OPT be a normalized optimal solution.

Let t_i be the time of RR's i^{th} rejection. For notational convenience, we define $t_0 = 0$. We show that the time between t_{i-1} and t_i is at most $c/(k+i)$. The result then follows from Lemma 1 since the i^{th} rejection occurs before $\tau_{k,i}$ as defined in Equation 2. Consider how the ratio $cost(RR, t)/cost(EOPT, t)$ changes between times t_{i-1} and t_i. All k extra jobs contribute to the denominator since they remain active in the pseudoschedule even if OPT rejects them. Only $k - (i-1) = k + 1 - i$ of them contribute to the numerator since $i - 1$ have already been rejected. In addition to the extra jobs, there is at least one other job active at all times or RR would work on extra jobs. This job and any other non-extra job contributes equally to the numerator and denominator since OPT and RR process jobs in the same order. Thus, the rejection occurs latest if $k + 2 - i$ jobs contribute to the numerator and $k + 1$ contribute to the denominator. Therefore, if RR has not made the i^{th} rejection before time $t_{i-1} + c/(k+i)$, the largest possible value of the ratio in Equation 3 is

$$\frac{cost(RR, t_{i-1}) + (k+2-i)c/(k+i) + c}{cost(EOPT, t_{i-1}) + (k+1)c/(k+i)} \leq 2$$

and RR makes the i^{th} rejection at that time.

It remains to show that each extra job is rejected within time c of its release when extra jobs arrive at several times. Suppose to the contrary that an input instance exists in which some job J is not completed or rejected within time c of its release. Since extra jobs arriving after job J only hasten its rejection, we assume the extra jobs arriving with J are the last extra jobs and the only ones not rejected within time c of their arrival. We also assume that non-extra jobs arrive one per unit time since deviations from this also hasten rejections. Call each set of a extra jobs arriving concurrently a *group*. We give a modification of the instance that reduces the number of groups in such a way that every rejection after the arrival of the second group occurs no earlier. This suffices since repeatedly applying this transformation to our initial instance creates an instance with a single group of extra jobs that are not all rejected within time c, which we have already shown cannot occur.

Suppose the first two groups of extra jobs arrive at times 0 and t, respectively. Let k be the size of the first group and let i be the number of these jobs rejected by time t. Our transformation removes all jobs arriving before time t and adds $k-i$ jobs to the second group. We show the rejections occur later in this modified instance by showing that $cost(RR,t)$ grows at least as quickly and $cost(EOPT,t)$ grows more slowly. The value of $cost(RR,t)$ grows at least as quickly at time t since exactly the same number of jobs are active in RR. The faster growth continues after time t because slower growth of $cost(EOPT,t)$ delays RR's rejections. To show that $cost(EOPT,t)$ grows more slowly, consider how the first group of extra jobs contribute to it after time t in the original instance. We showed above that an extra group with k jobs causes i rejections by time $\sum_{j=1}^{i} c/(k+j) \le ci/k$ after its arrival. Thus, $t \le ci/k$. Since the first group has k jobs contributing to $cost(EOPT,t)$ until time c, their total contribution after time t is at least $ck(1-i/k) = c(k-i)$. This is exactly the total contribution of the $k-i$ jobs we replaced them with. In addition, the replacement jobs contribute until time $t+c$ rather than stopping at time c. Since the contribution of the replacement jobs is no larger and it is more dispersed, $cost(EOPT,t)$ grows more slowly. □

4 NP-completeness of minimizing flow with rejections

Turning to the offline problem, we show that a restricted version of the problem is NP-complete.

THEOREM 10 *Minimizing total flow time with rejections is NP-complete even when the cost of rejecting a job is proportional to its processing time.*

Note that this theorem implies that minimizing the sum of completion times with rejection is also NP-complete since the same schedule is optimal for total flow time and sum of completion times. In fact, our proof uses ideas of Du et al. [5], who proved that it is NP-complete to find a multiprocessor preemptive schedule minimizing the sum of completion times. Rejected jobs in our schedule are equivalent to jobs running on a second processor in theirs.

Proof: The problem is in NP because once the jobs to reject are specified, the others are run using SRPT [2]. To show hardness, we give a reduction from PARTITION [7]:

> PARTITION: Given a set $A = \{a_1, a_2, \ldots, a_n\}$ of positive integers, does there exist a partition of A into A_1 and A_2 such that $\sum_{a_i \in A_1} a_i = \sum_{a_i \in A_2} a_i$?

Let $B = \sum_{i=1}^{n} a_i$. We assume B is a multiple of 2 since otherwise a partition cannot exist. We also assume $n \ge 5$ since small instances of PARTITION can be quickly decided. We create a scheduling instance with $n+2$ jobs. For each element $a_i \in A$, we create job J_i with $r_i = 2(i-1)B$, $p_i = 2a_i$, and $c_i = 5a_i$. In addition, there are two jobs J_{n+1} and J_{n+2} with $r_{n+1} = 0$, $p_{n+1} = 2nB$, $c_{n+1} = 5nB$, $r_{n+2} = 2nB + B$, $p_{n+2} = 2nB$, and $c_{n+2} = 5nB$. Observe that the rejection cost for each job is 5/2 times its processing time.

We claim that a partition exists if and only if a schedule exists with cost at most $(4n + 9/2)B$. Suppose a partition of A into A_1 and A_2 exists. Let S_1 and S_2 be sets of jobs containing the jobs corresponding to A_1 and A_2 respectively. We create a schedule that rejects the jobs in S_1 as they arrive and runs the jobs in S_2 to completion as they arrive. In the times between these executions, it runs J_{n+1}. Since the jobs in S_2 have total execution time $\sum_{J_i \in S_2} p_i = \sum_{a_i \in A_2} 2a_i = B$, job J_{n+1} finishes at time $2nB + B$. Thus, we can run job J_{n+2} as soon as it arrives. This schedule has total cost $\sum_{J_i \in S_2} p_i + \sum_{J_i \in S_1} c_i + (2nB + B) + 2nB = B + (5/2)B + 4nB + B = (4n + 9/2)B$.

Now suppose a schedule with cost at most $(4n+9/2)B$ exists. Clearly, neither J_{n+1} nor J_{n+2} are rejected. Let S_1 be the set of rejected jobs and S_2 be the other members of $\{J_1, J_2, \ldots, J_n\}$. Also, let A_1 and A_2 contain the a_i corresponding to members of S_1 and S_2 respectively. We assume that each member J_i of S_2 is run immediately following its arrival since otherwise doing so and running J_{n+1} when J_i was run produces a schedule with no greater cost. Similarly, we assume that job J_{n+1} is completed before J_{n+2} is started. If $\sum_{a_i \in A_2} a_i > B/2$, then J_{n+1} does not complete before the arrival of J_{n+2} and so the cost is at least $5\sum_{a_i \in A_1} a_i + 2\sum_{a_i \in A_2} a_i + \left(2nB + 2\sum_{a_i \in A_2} a_i\right) + \left(2nB + 2\sum_{a_i \in A_2} a_i - B\right) = 4nB + 4B + \sum_{a_i \in A_2} a_i > (4n + 9/2)B$. (Recall that $\sum_{a_i \in A_1} a_i + \sum_{a_i \in A_2} a_i = B$.) However, if $\sum_{a_i \in A_2} a_i < B/2$, then the cost is at least $5\sum_{a_i \in A_1} a_i + 2\sum_{a_i \in A_2} a_i + \left(2nB + 2\sum_{a_i \in A_2} a_i\right) + 2nB = 4nB + 5B - \sum_{a_i \in A_2} a_i > (4n + 9/2)B$. Thus, $\sum_{a_i \in A_2} a_i = B/2 = \sum_{a_i \in A_1} a_i$ so A_1 and A_2 form a partition of A. □

5 Discussion

Although we have made progress on several parts of this problem, there remain areas for further work. We are most interested in closing the gap between the lower bound of $e/(e-1) \approx 1.582$ and the various 2-competitive algorithms for unit-length jobs with rejection cost c. We believe that RR can be better than 2-competitive; certainly there is slack for a single group of extra jobs, which $RR(2)$ finishes by approximately $0.7c$. There is also still a gap between our lower bound of $(2 + \sqrt{2})/2 \approx 1.707$ and the 2-competitive algorithm of Bansal et al. [2] for jobs with arbitrary processing times and rejection cost c. In addition, it would be interesting to know if randomized algorithms can beat the competitive ratios given in Theorems 3 and 4. In the offline setting, we have shown that one form of the problem is NP-complete, but it is not known whether the problem remains NP-complete when all jobs have rejection cost c. Another open question is whether either the full problem or this special case admit a PTAS, i.e. a $(1 + \epsilon)$-approximation algorithm that runs in polynomial-time for any constant ϵ.

More generally, much more can be done to address the question of how to combine planning with scheduling. This question has not been considered in many scheduling settings nor has anyone characterized the requirements of a specific application.

Acknowledgments

The author thanks Dan Cranston for help solving the recurrence giving the lower bound $e/(e-1)$. He also thanks Cindy Phillips, Vitus Leung, Kevin Milans, John Fischer, and the anonymous referees for helpful comments about revising the paper.

References

[1] N. Bansal. *Algorithms for flow time scheduling.* PhD thesis, Carnegie Mellon University, 2003. `http://www.research.ibm.com/people/n/nikhil/papers/thesis.pdf`.

[2] N. Bansal, A. Blum, S. Chawla, and K. Dhamdhere. Scheduling for flow-time with admission control. In *Proc. 11th Annual European Symp. Algorithms*, number 2832 in LNCS, pages 43–54, 2003.

[3] N. Bansal and K. Dhamdhere. Minimizing weighted flow time. In *Proc. 14th Annual ACM-SIAM Symp. Discrete Algorithms*, pages 508–516, 2003.

[4] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, and L. Stougie. Multiprocessor scheduling with rejection. In *Proc. 7th Annual ACM-SIAM Symp. Discrete Algorithms*, pages 95–103, 1996.

[5] J. Du, J.Y.-T. Leung, and G.H. Young. Minimizing mean flow time with release time constraints. *Theoretical Computer Science*, 75(3):347–355, 1990.

[6] D. Engels, D. Karger, S. Kolliopoulos, S. Sengupta, R. Uma, and J. Wein. Techniques for scheduling with rejection. In *Proc. 6th Annual European Symp. Algorithms*, volume 1461 of *LNCS*, pages 490–501, 1998.

[7] M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman and Company, 1979.

[8] H. Hoogeveen, M. Skutella, and G. Woeginger. Preemptive scheduling with rejection. In *Proc. 8th Annual European Symp. Algorithms*, volume 1879 of *LNCS*, pages 268–277, 2000.

[9] I.M. Ovacik and R. Uzsoy. *Decomposition methods for complex factory scheduling problems*. Kluwer Academic Publishers, 1997.

[10] K. Pruhs, J. Sgall, and E. Torng. Online scheduling. In J.Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 15, pages 15–1–15–41. CRC Press, 2004.

[11] W. Rudin. *Principles of mathematical analysis*. McGraw-Hill, Inc., 3rd edition, 1976.

[12] L. Schrage. A proof of the optimality of the shortest processing remaining time discipline. *Operations Research*, 16:687–690, 1968.

[13] S.S. Seiden. Preemptive multiprocessor scheduling with rejection. *Theoretical Computer Science*, 262:437–458, 2001.

[14] S. Sengupta. Algorithms and approximation schemes for minimum lateness/tardiness scheduling with rejection. In *Proc. 9th Workshop on Algorithms and Data Structures*, number 2748 in LNCS, pages 79–90, 2003.

[15] D. Smith. A new proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 26(1):197–199, 1976.

Appendix: Proof of Lemmas 1 and 2

Before proving our main results, we show the following technical lemma:

LEMMA 11 *Consider the axis-aligned rectangle with corners* $(i, 1/i)$ *and* $(i+1, 1/(i+1))$. *The proportion of its area above the function* $f(x) = 1/x$ *is at most* $1/2 + 1/(2i)$.

Proof: The area of the box is $1/i - 1/(i+1) = 1/(i(i+1))$. The area within the box and under the curve is

$$\int_i^{i+1} \frac{dx}{x} - \frac{1}{i+1} = \ln \frac{i+1}{i} - \frac{1}{i+1}$$

Thus, the proportion of area under the curve is $1 - i(i+1)\ln((i+1)/i) + i = 1 + i - i(i+1)\ln(1+1/i)$. Using the first two terms of the Taylor series expansion for $\ln(1+x)$, this is at most

$$1 + i - (i^2 + i)\left(\frac{1}{i} - \frac{1}{2i^2}\right) = 1/2 + 1/(2i)$$

□

This allows us to prove the first of our lemmas about $\tau_{k,k}$.

Proof of Lemma 1: For the value of $\tau_{k,k}$, we use Equation 2:

$$\tau_{k,k} = \sum_{j=1}^{k} \frac{1}{(k+1)/(e-1) + j - 1} = \int_1^{k+1} \frac{dx}{(k+1)/(e-1) + \lfloor x \rfloor - 1}$$

We want to remove the floor from this integral, but doing so decreases the value. We bound the difference by considering the total area in the axis-aligned rectangles with corners along the curve

$y = 1/x$ at $x = (k+1)/(e-1) + i$ for $i = 0, 1, \ldots, k$. These boxes have total area

$$\sum_{i=(k+1)/(e-1)}^{(k+1)/(e-1)+k} \left(\frac{1}{i} - \frac{1}{i+1} \right) = \frac{e-1}{k+1} - \frac{e-1}{ke+1}$$

When $k \geq 10$, the proportion of this area that is part of the error is at most

$$\frac{1}{2} + \frac{1}{2i} \leq \frac{1}{2} + \frac{e-1}{22} = \frac{e+10}{22} \leq \frac{1}{e-1}$$

by Lemma 11. Thus, $\tau_{k,k}$ for $k \geq 10$ obeys

$$\begin{aligned}
\tau_{k,k} &\leq \int_1^{k+1} \frac{dx}{(k+1)/(e-1) + x - 1} + \frac{1}{e-1} \left(\frac{e-1}{k+1} - \frac{e-1}{ke+1} \right) \\
&= \int_1^{k+1} \frac{(e-1)dx}{k+2-e+x(e-1)} + \frac{k(e-1)}{(k+1)(ke+1)} \\
&< \ln(k+2-e+(k+1)(e-1)) - \ln(k+2-e+(e-1)) + \frac{e-1}{ke+1} \\
&= \ln \frac{ke+1}{k+1} + \frac{e-1}{ke+1} = 1 - \ln \frac{ke+e}{ke+1} + \frac{e-1}{ke+1} \\
&= 1 - \ln \left(1 + \frac{e-1}{ke+1} \right) + \frac{e-1}{ke+1}
\end{aligned}$$

The last line is at most 1 since $\ln(1+x) = \sum_{i=1}^{\infty} (-1)^{i+1} x^i / i$, which is greater than x for $x < 1$.

For $k < 10$, we verify the identity by hand using Equation 2: $\tau_{1,1} \approx 0.859$, $\tau_{2,2} \approx 0.937$, $\tau_{3,3} \approx 0.961$, $\tau_{4,4} \approx 0.972$, $\tau_{5,5} \approx 0.979$, $\tau_{6,6} \approx 0.983$, $\tau_{7,7} \approx 0.985$, $\tau_{8,8} \approx 0.987$, $\tau_{9,9} \approx 0.989$. □

The other observation follows quickly from this:

Proof of Lemma 2:

$$\begin{aligned}
\tau_{k,k} &= \sum_{j=1}^{k} \frac{1}{(k+1)/(e-1) + j - 1} \\
&\geq \sum_{j=1}^{k} \frac{1}{\lceil (k+1)/(e-1) \rceil + j - 1} \\
&= H_{\lceil (k+1)/(e-1) \rceil + k - 1} - H_{\lceil (k+1)/(e-1) \rceil - 1}
\end{aligned}$$

where $H_n = \sum_{j=1}^{n} 1/n$ is the n^{th} *Harmonic number*. Recall that $\lim_{n\to\infty}(H_n - \ln n) = \gamma$, where $\gamma \approx 0.577$ is *Euler's constant* [11]. Thus,

$$\begin{aligned}
\lim_{k\to\infty} \tau_{k,k} &\geq \lim_{k\to\infty} \left(H_{\lceil (k+1)/(e-1) \rceil + k - 1} - H_{\lceil (k+1)/(e-1) \rceil - 1} \right) \\
&= \lim_{k\to\infty} \left(\ln((k+1)/(e-1) + k - 1) - \ln((k+1)/(e-1) - 1) \right) \\
&= \lim_{k\to\infty} \ln \frac{(k+1)/(e-1) + k - 1}{(k+1)/(e-1) - 1} = \ln(1 + e - 1) \\
&= 1
\end{aligned}$$

The result then follows from Lemma 1. □