Scheduling for Flow-Time with Admission Control*

Nikhil Bansal[1], Avrim Blum[1], Shuchi Chawla[1], and Kedar Dhamdhere[1]

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA. {nikhil,avrim,shuchi,kedar}@cs.cmu.edu

Abstract. We consider the problem of scheduling jobs on a single machine with preemption, when the server is allowed to reject jobs at some penalty. We consider minimizing two objectives: total flow time and total job-idle time (the idle time of a job is the flow time minus the processing time). We give 2-competitive online algorithms for the two objectives and extend some of our results to the case of weighted flow time and machines with varying speeds. We also give a resource augmentation result for the case of arbitrary penalties achieving a competitive ratio of $O(\frac{1}{\epsilon}(\log W + \log C)^2)$ using a $(1+\epsilon)$ speed processor. Finally, we present a number of lower bounds for both the case of uniform and arbitrary penalties.

1 Introduction

Consider a large distributed system with multiple machines and multiple users who submit jobs to these machines. The users want their jobs to be completed as quickly as possible, but they may not have exact knowledge of the current loads of the processors, or the jobs submitted by other users in the past or near future. However, let us assume that each user has a rough estimate of the typical time she should expect to wait for a job to be completed. One natural approach to such a scenario is that when a user submits a job to a machine, she informs the machine of her estimate of the waiting time if she were to send the job elsewhere. We call this quantity the *penalty* of a job. The machine then might service the job, in which case the cost to the user is the flow time of the job (the time elapsed since the job was submitted). Or else the machine might reject the job, possibly after the job has been sitting on its queue for some time, in which case the cost to the user is the penalty of the job plus the time spent by the user waiting on this machine so far.

To take a more human example, instead of users and processors, consider journal editors and referees. When an editor sends a paper to a referee, ideally she would like a report within some reasonable amount of time. Less ideally, she would like an immediate response that the referee is too busy to do it. But even worse is a response of this sort that comes 6 months later after the referee originally agreed to do the report. However,

* This research was supported in part by NSF grants CCR-0105488, NSF-ITR CCR-0122581, NSF-ITR IIS-0121678, and an IBM Graduate Fellowship.

from the referee's point of view, it might be that he thought he would have time when he received the request, but then a large number of other tasks arrived and saying no to the report (or to some other task) is needed to cut his losses.

Motivated by these scenarios, in this paper we consider this problem from the point of view of a single machine (or researcher/referee) that wants to be a good sport and minimize the total cost to users submitting jobs to that machine. That is, it wants to minimize the total time jobs spend on its "to-do list" (flow time) plus rejection penalties.[1]

Specifically, we consider the problem of scheduling on a single machine to minimize flow time (also job-idle time) when jobs can be rejected at some cost. Each job j has a release time r_j, a processing time p_j, and we may at any time cancel a job at cost c_j. For most of the paper, we focus on the special case that the cancellation costs are all equal to some fixed value c — even this case turns out to be nontrivial — though we give some results for general c_j as well. In this paper, we consider clairvoyant algorithms, that is, whenever a job is released, its size and penalty is revealed.

In the flow-time measure, we pay for the total time a job is in the system. So, if a job arrives at time 1 and we finish it by time 7, we pay 6 units. If we choose to cancel a job, the cancellation cost is added on. Flow-time is equivalent to saying that at each time step, we pay for the number of jobs currently in the system (i.e., the current size of the machine's to-do list). In the job-idle time measure, we pay at each time step for the number of jobs currently in the system minus one (the one we are currently working on), or zero if there are no jobs in the system. Because job idle time is smaller than flow time, it is a strictly harder problem to approximate, and can even be zero if jobs are sufficiently well-spaced. Preemption is allowed, so we can think of the processor as deciding at each time step how it wants to best use the next unit of time. Note that for the flow-time measure, we can right away reject jobs that have size more than c, because if scheduled, these add at least c to the flow-time. However, this is not true for the job-idle time measure.

To get a feel for this problem, notice that we can model the classic ski-rental problem as follows. Two unit-size jobs arrive at time 0. Then, at each time step, another unit-size job arrives. If the process continues for less than c time units, the optimal solution is not to reject any job. However, if it continues for c or more time units, then it would be optimal to reject one of the two jobs at the start. In fact, this example immediately gives a factor 2 lower bound for deterministic algorithms for job-idle time, and a factor $3/2$ lower bound for flow time.

To get a further feel for the problem, consider the following online algorithm that one might expect to be constant-competitive, but in fact does not work: Schedule jobs using the Shortest Remaining Processing Time (SRPT) policy (the optimal algorithm when rejections are not allowed), but whenever a job has been in the system for more than c time units, reject this job, incurring an additional c cost. Now consider the

[1] However, to be clear, we are ignoring issues such as what effect some scheduling policy might have on the rest of the system, or how the users ought to behave, etc.

behavior of this algorithm on the following input: m unit size jobs arrive at time 0, where $m < c$, and subsequently one unit size job arrives in every time step for n steps. SRPT (breaking ties in favor of jobs arriving earlier) will schedule every job within m time units of its arrival. Thus, the proposed algorithm does not reject any job, incurring a cost of mn, while *Opt* rejects $m-1$ jobs in the beginning, incurring a cost of only $n+(m-1)c$. This gives a competitive ratio of m as $n \to \infty$.
A complaint one might have about the job-idle time measure is that it gives the machine credit for time spent processing jobs that are later rejected. For example, if we get a job at time 0, work on it for 3 time units, and reject it at time 5, we pay $c+2$ rather than $c+5$. A natural alternative would be to define the cost so that no credit is given for time spent processing jobs that end up getting rejected. Unfortunately, that definition makes it impossible to achieve any finite competitive ratio. In particular, if a very large job arrives at time 0, we cannot reject it since it may be the only job and OPT would be 0; but, then if unit-size jobs appear at every time step starting at time tc, we have committed to cost tc whereas OPT could have rejected the big job at the start for a cost of only c.
The main results of this paper are as follows: In section 2, we give a 2-competitive online algorithm for flow time and job-idle time with penalty. Note that, for job-idle time, this matches the simple lower bound given above. The online algorithm is extended to an $O(\log^2 W)$ algorithm for *weighted* flow time in Section 3, where W is the ratio between the maximum and minimum weight of any job. In Section 4 we give lower bounds for the problem with arbitrary rejection penalties and also give a $O(\frac{1}{\epsilon}(\log W + \log C)^2)$ competitive algorithm using a $(1+\epsilon)$ speed processor in the resource augmentation model, where C is the ratio between the maximum and the minimum penalty for any job.

1.1 Related Previous Work

Flow time is a widely used criterion for measuring performance of scheduling algorithms. For the unweighted case, it has been long known [1] that the Shortest Remaining Processing Time (SRPT) policy is optimal for this problem. The weighted problem is known to be much harder. Recently Chekuri et al [2, 3] gave the first non trivial semi-online algorithm for the problem that achieves a competitive ratio of $O(\log^2 P)$. Here P is the ratio of the maximum size of any job to the minimum size of any job. Bansal et al [4] give another online algorithm achieving a ratio of $O(\log W)$, and a semi-online algorithm which is $O(\log n + \log P)$ competitive. Also related is the work of Becchetti et al [5], who give a $(1+1/\epsilon)$ competitive algorithm for weighted flow time using a $(1+\epsilon)$ speed processor.
Admission control has been studied for a long time in circuit routing problems (see, e.g., [6]). In these problems, the focus is typically on approximately maximizing the throughput of the network. In scheduling problems, the model of rejection with penalty was first introduced by Bartal et al [7]. They considered the problem of minimizing makespan on multiple machines with rejection and gave a $1+\phi$ approximation for

the problem where ϕ is the golden ratio. Variants of this problem have been subsequently studied by [8, 9]. Seiden [8] extends the problem to a pre-emptive model and improves the ratio obtained by [7] to 2.38.

More closely related to our work is the model considered by Engels et al [10]. They consider the problem of minimizing weighted *completion* time with rejections. However, there are some significant differences between their work and ours. First, their metric is different. Second, they only consider the offline problem and give a constant factor approximation for a special case of the problem using LP techniques.

1.2 Notation and Definitions

We consider the problem of online pre-emptive scheduling of jobs so as to minimize flow time with rejections or job idle time with rejections. Jobs arrive online; their processing time is revealed as they arrive. A problem instance $\mathcal{J}$ consists of n jobs and a penalty c. Each job j is characterized by its release time r_j and its processing time p_j. P denotes the ratio of the maximum processing time to the minimum processing time.

At any point of time an algorithm can schedule or reject any job released before that time. For a given schedule S, at any time t, a job is called active if it has not been finished or rejected yet. The completion time κ_j of a job is the time at which a job is finished or rejected.

The flow time of a job is the total time that the job spends in the system, $f_j = \kappa_j - r_j$. The flow time of a schedule S denoted by $F(S)$ is the sum of flow times of all jobs. Similarly, the job idle time of a job is the total time that the job spends in queue not being processed. This is $f_j - p_j$ if the job is never rejected, or f_j-(the duration for which it was scheduled) otherwise. The job idle time of a schedule denoted by $I(S)$ is the sum of job idle times of all jobs. For a given algorithm A, let R_A be the set of jobs that were rejected and let S_A be the schedule produced. Then, the flow time with rejections of the algorithm is given by $F(S_A)+c|R_A|$. Similarly the job idle time with rejections of the algorithm is given by $I(S_A) + c|R_A|$.

We use A to denote our algorithms and the cost incurred by them. We denote the Optimal algorithm and its cost by Opt.

In the weighted problem, every job has a weight w_j associated with it. Here, the objective is to minimize weighted flow time with rejections. This is given by $\sum_j (w_j f_j) + c|R_A|$. W denotes the ratio of weights of the highest weight class and the least weight class. As in the unweighted case, the weight of a job is revealed when the job is released.

We also consider the case when different jobs have different penalties. In this case, we use c_j to denote the penalty of job j. c_{max} denotes the maximum penalty and c_{min} the minimum penalty. We use C to denote the ratio c_{max}/c_{min}.

Our algorithms do not assume knowledge of C, W or P. Finally, by a stream of jobs of size x, we mean a string of jobs each of size x, arriving every x units of time.

1.3 Preliminaries

We first consider some properties of the optimal solution (*Opt*) which will be useful in deriving our results.

Fact 1 *If Opt rejects a job j, it is rejected the moment it arrives.*

Fact 2 *Given the set of jobs that Opt rejects, the remaining jobs must be serviced in Shortest Remaining Processing Time (SRPT) order.*

Fact 3 *In the uniform penalty model, if a job j is rejected, then it must be the job that currently has the largest remaining time.*

2 An Online Algorithm

In this section, we will give online algorithms for minimizing flow time and job idle time with rejections.

2.1 Minimizing Flow Time

Flow time of a schedule can be expressed as the sum over all time steps of the number of jobs in the system at that time step. Let ϕ be a counter that counts the flow time accumulated until the current time step. The following algorithm achieves 2-competitiveness for flow time with rejections:

The Online Algorithm. Starting with $\phi = 0$, at every time step, increment ϕ by the number of active jobs in the system at that time step. Whenever ϕ crosses a multiple of c, reject the job with the largest remaining time. Schedule active jobs in SRPT order.

Let the schedule produced by the above algorithm be S and the set of rejected jobs be R.

Lemma 1. *The cost of the algorithm is $\leq 2\phi$.*

Proof. This follows from the behavior of the algorithm. In particular, $F(S)$ is equal to the final value in the counter ϕ, and the total rejection cost $c|R|$ is also at most ϕ because $|R|$ increases by one (a job is rejected) every time ϕ gets incremented by c.

The above lemma implies that to get a 2-approximation, we only need to show that $\phi \leq Opt$. Let us use another counter ψ to account for the cost of Opt. We will show that the cost of Opt is at least ψ and at every point of time $\psi \geq \phi$. This will prove the result.
The counter ψ works as follows: Whenever Opt rejects a job, ψ gets incremented by c. At other times, if $\phi = \psi$, then ϕ and ψ increase at the same rate (i.e. ψ stays equal to ϕ). At all other times ψ stays constant. By design, we have the following:

Fact 4 *At all points of time, $\psi \geq \phi$.*

Let $k = \lfloor \frac{\psi}{c} \rfloor - \lfloor \frac{\phi}{c} \rfloor$. Let n_o and n_a denote the number of active jobs in Opt and A respectively. Arrange and index the jobs in Opt and A in the order of decreasing remaining time. Let us call the k longest jobs of A *marked*. We will now prove the following:

Lemma 2. *At all times $n_o \geq n_a - k$.*

Lemma 2 will imply $Opt \geq \psi$ (and thus, 2-competitiveness) by the following argument: Whenever ψ increases by c, Opt spends the same cost in rejecting a job. When ψ increases at the same rate as ϕ, we have that $\psi = \phi$. In this case $k = 0$ and thus Opt has at least as many jobs in system as the online algorithm. Since the increase in ϕ (and thus ψ) accounts for the flow time accrued by the online algorithm, this is less than the flow time accrued by Opt. Thus the cost of Opt is bounded below by ψ and we are done.
We will prove Lemma 2 by induction over time. For this we will need to establish a suffix lemma. We will ignore the marked jobs while forming suffixes.
Let $P_o(i)$ (called a suffix) denote the sum of remaining times of jobs $i, \ldots, n_o$ in Opt. Let $P_a(i)$ denote the sum of remaining times of jobs $i+k, \ldots, n_a$ in A ($i, \ldots, n_a - k$ among the unmarked jobs). For instance, Figure 1 below shows the suffices for $i = 2$ and $k = 2$.

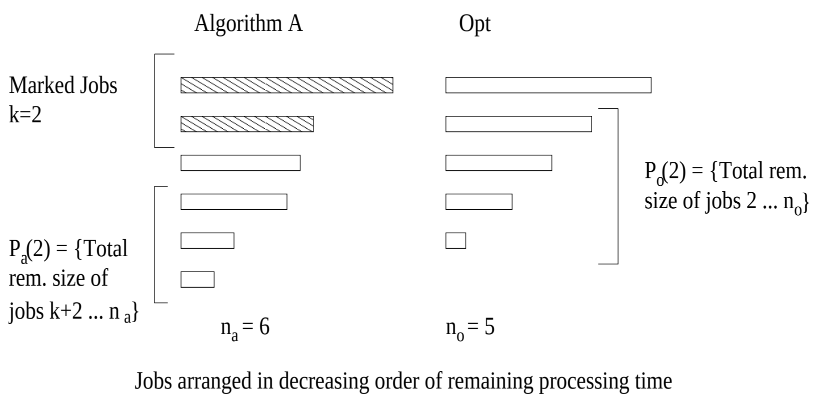

Fig. 1. *Notation used in proof of Theorem 1*

Lemma 3. *At all times, for all i, $P_a(i) \leq P_o(i)$.*

Proof. (of Lemma 2 using Lemma 3) Using $i = n_a - k$, we have $P_o(n_a - k) \geq P_a(n_a - k) > 0$. Therefore, $n_o \geq n_a - k$.

Proof. (Lemma 3) We prove the statement by induction over the various events in the system. Suppose the result holds at some time t. First consider the simpler case of no arrivals. Furthermore, assume that the value of k does not change from time t to $t+1$. Then, as A always works on the job n_a, $P_a(i)$ decreases by 1 for each $i \leq n_a - k$ and by 0 for $i > n_a - k$. Since $P_o(i)$ decreases by at most 1, the result holds for this case.
If the value of k changes between t and $t+1$, then since there are no arrivals (by assumption), it must be the case that A rejects some job(s) and k decreases. However, note that rejection of jobs by A does not affect any suffix under A (due to the way $P_a(i)$ is defined). Thus the argument in the previous paragraph applies to this case.
We now consider the arrival of a job J at time t. If J is rejected by Opt, the suffixes of Opt remain unchanged and the value of k increases by 1. If J gets marked under A, none of the suffixes under A change either, and hence the invariant remains true. If J does not get marked, some other job with a higher remaining time than J must get marked. Thus the suffixes of A can only decrease.
If J is not rejected by Opt, we argue as follows: Consider the situation just before the arrival of J. Let C be the set of unmarked jobs under A and D the set of all jobs under Opt. On arrival of J, clearly J gets added to D. If J is unmarked under A it gets added to C else if it gets marked then a previously marked job $J' \in A$, with a smaller remaining time than J gets added to C. In either case, the result follows from Lemma 4 (see Proposition A.7, Page 120 in [11] or Page 63 in [12]), which is a result about suffixes of sorted sequences, by setting $\mathcal{C} = C$, $\mathcal{D} = D$, $d' = J$ and $c' = J$ or J'.

Lemma 4. *Let $\mathcal{C} = \{c_1 \geq c_2 \geq \ldots\}$ and $\mathcal{D} = \{d_1 \geq d_2 \geq \ldots\}$ be sorted sequences of non-negative numbers. We say that $\mathcal{C} \prec \mathcal{D}$ if $\sum_{j \geq i} c_j \leq \sum_{j \geq i} d_i$ for all $i = 1, 2, \ldots$. Let $\mathcal{C} \cup \{c'\}$ be the sorted sequence obtained inserting c' in $\mathcal{C}$. Then, $\mathcal{C} \prec \mathcal{D}$ and $c' \leq d' \Rightarrow \mathcal{C} \cup \{c'\} \prec \mathcal{D} \cup \{d'\}$.*

Thus we have the following theorem:

Theorem 1. *The above online algorithm is 2-competitive with respect to Opt for the problem of minimizing flow time with rejections.*

2.2 Minimizing Job Idle Time

Firstly note that the job idle time of a schedule can by computed by adding the contribution of the jobs waiting in the queue (that is, every job except the one that is being worked upon, contributes 1) at every time step.
The same online algorithm as in the previous case works for minimizing job idle time with the small modification that the counter ϕ now increments by the number of waiting jobs at every time step. The analysis is similar and gives us the following theorem:

Theorem 2. *The above online algorithm is 2-competitive with respect to Opt for the problem of minimizing job idle time with rejections.*

2.3 Varying Server Speeds

For a researcher managing his/her to-do list, one typically has different amounts of time available on different days. We can model this as a processor whose speed changes over time in some unpredictable fashion (i.e., the online algorithm does not know what future speeds will be in advance). This type of scenario can easily fool some online algorithms: e.g., if the algorithm immediately rejected any job of size $\geq c$ according to the current speed, then this would produce an unbounded competitive ratio if the processor immediately sped up by a large factor.
However, our algorithm gives a 2-approximation for this case as well. The only effect of varying processor speed on the problem is to change sizes of jobs as time progresses. Let us look at the problem from a different angle: the job sizes stay the same, but time moves at a faster or slower pace. The only effect this has on our algorithm is to change the time points at which we update the counters ϕ and ψ. However, notice that our algorithm is locally optimal: at all points of time the counter ψ is at most the cost of *Opt*, and $\phi \leq \psi$, irrespective of whether the counters are updated more or less often. Thus the same result holds.

2.4 Lower Bounds

We now give a matching lower bound of 2 for waiting time and 1.5 for flow time, on the competitive ratio of any deterministic online algorithm. Consider the following example: Two jobs of size 1 arrive at $t = 0$. The adversary gives a stream of unit size jobs starting at $t = 1$ until the algorithm rejects a job.
Let x be the time when the algorithm first rejects a job. In the waiting time model, the cost of the algorithm is $x + c$. The cost of the optimum is $\min(c, x)$, since it can either reject a job in the beginning, or not reject at all. Thus we have a competitive ratio of 2.
The same example gives a bound of 1.5 for flow time. Note that the cost of the online algorithm is $2x + c$, while that of the optimum is $\min(x + c, 2x)$.

Theorem 3. *No online algorithm can achieve a competitive ratio of less than 2 for minimizing waiting time with rejections or a competitive ratio of less than 1.5 for minimizing flow time with rejections.*

3 Weighted flow time with weighted penalties

In this section we consider the minimization of weighted flow time with admission control. We assume that each job has a weight associated with it. Without loss of generality, we can assume that the weights are powers of 2. This is because rounding up the weights to the nearest power of 2 increases the competitive ratio by at most a factor of 2. Let $a_1, a_2, \ldots, a_k$ denote the different possible weights, corresponding to weight classes $1, 2, \ldots, k$. Let W be the ratio of maximum to minimum weight. Then, by our assumption, k is at most $\log W$. We will consider the following two models for penalty. The general case of arbitrary penalties is considered in the next section.

Uniform penalty: Jobs in each weight class have the same penalty c of rejection.

Proportional penalty: Jobs in weight class j have rejection penalty $a_j c$.

For both these cases, we give an $O(\log^2 W)$ competitive algorithm. This algorithm is based on the Balanced SRPT algorithm due to Bansal *et al.* [4]. We modify their algorithm to incorporate admission control. The modified algorithm is described below.

Algorithm Description: As jobs arrive online, they are classified according to their weight class. Consider the weight class that has the minimum total remaining time of jobs. Ties are resolved in favor of higher weight classes. At each time step, we pick the job in this weight class with smallest remaining time and schedule it.

Let ϕ be a counter that counts the total weighted flow time accumulated until current time step. For each weight class j, whenever ϕ crosses the penalty c (resp. $a_j c$), we reject a job with the largest remaining time from this class.

Analysis: We will imitate the analysis of the weighted flow time algorithm. First we give an upper bound on the cost incurred by the algorithm. Let $F(S)$ be the final value of counter ϕ. The cost of rejection, $c|R|$, is bounded by $k\phi$, because rejections $|R_j|$ in weight class j increase by 1 every time ϕ increases by c_j. Thus we have,

Lemma 5. *The total cost of the algorithm is* $\leq (k+1)\phi$

In order to lower bound the cost of optimal offline algorithm, we use a counter ψ. The counter ψ works as follows: Whenever *Opt* rejects a job of weight class j, ψ gets incremented by c_j. At other times, if $\phi = \psi$, then ϕ and ψ increase at the same rate (i.e. ψ stays equal to ϕ), otherwise, ψ stays constant. By design, we have the following:

Fact 5 *At all points of time,* $\psi \geq \phi$.

Now we show that ψ is a lower bound on $k \cdot Opt$. Let $m_j = \lfloor \frac{\psi}{kc_j} \rfloor - \lfloor \frac{\phi}{kc_j} \rfloor$. In both *Opt* and our algorithm, arrange active jobs in each weight class in decreasing order of remaining processing time. We call the first m_j jobs of weight class j in our algorithm as marked. Now ignoring the marked jobs, we can use theorem 2 from Bansal *et al.* [4]. We get the following:

Lemma 6. *The total weight of unmarked jobs in our algorithm is no more than k times the total weight of jobs in Opt.*

Proof. (*Sketch*) The proof follows along the lines of lemma 2 in Bansal *et al.* [4]. Their proof works in this case if we only consider the set of unmarked jobs in our algorithm. However, due to rejections, we need to check a few more cases.

We first restate their lemma in terms suitable for our purpose. Let $B(j,l)$ and $P(j,l)$ denote a *prefix* of the jobs in our algorithm and *Opt* algorithm respectively. Then, we define the suffixes $\overline{B}(j,l) = J_a - B(j,l)$ and $\overline{P}(j,l) = J_o - P(j,l)$, where J_a and J_o are the current sets of jobs in our algorithm and the *Opt* algorithm respectively.

Lemma 7. ([4]) *The total remaining time of the jobs in the suffix* $\overline{B}(j,l)$ *is smaller than the total remaining time of the jobs in* $\overline{P}(j,l)$.

We now consider the cases that are not handled by Bansal *et al.*'s proof. If a job of weight class j arrives and *Opt* rejects it, then the set of jobs with *Opt* does not change. On the other hand, m_j increases by at least 1. In our algorithm, if the new job is among top m_j jobs in its weight class, then it is marked and set of unmarked jobs remains the same. If the new job does not get marked, the suffixes of our algorithm can only decrease, since some other job with higher remaining time must get marked.
Similarly, when our algorithm rejects a job of class j, then the number of marked jobs m_j reduces by 1. However, the rejected job had highest remaining time in the class j. Hence none of the suffixes change.
Thus, we have established that the suffixes in our algorithm are smaller than the corresponding suffixes in the *Opt* algorithm at all times. The argument from Theorem 2 in [4] gives us the result that weight of unmarked jobs in our algorithm is at most $k \cdot Opt$.

To finish the argument, note that when the *Opt* algorithm rejects a job of weight class j, *Opt* increases by c_j. And ψ increases by kc_j. On the other hand, when ψ and ϕ increase together, we have $\psi = \phi$. There are no marked jobs, since $m_j = 0$ for all j. The increase in ψ per time step is same as the weight of all jobs in our algorithm. As we saw in the Lemma 6, this is at most k times the total weight of jobs in *Opt* algorithm. Thus, the total increase in ψ is bounded by $k \cdot Opt$.
In conjunction with Lemma 5, this gives us $O(\log^2 W)$ competitiveness.

4 Weighted Flow time with Arbitrary Penalties

In this section we will consider the case when different jobs have different weights and different penalties of rejection. First we will show that even for the simpler case of minimizing unweighted flow time with two different penalties, no algorithm can obtain a competitive ratio of less than $n^{\frac{1}{4}}$ or less than $C^{\frac{1}{2}}$. A similar bound holds even if there are two different penalties and the arrival times of high penalty jobs are known in advance. Then we will give an online algorithm that achieves a competitive ratio of $O(\frac{1}{\epsilon}(\log W + \log C)^2)$ using a processor of speed $(1+\epsilon)$.

4.1 Lower Bounds

Theorem 4. *For the problem of minimizing flow time or job idle time with rejection, and arbitrary penalties, no (randomized) online algorithm can achieve a competitive ratio of less than* $n^{\frac{1}{4}}$ *or* $C^{\frac{1}{2}}$. *Even when there are only two different penalties and the algorithm has knowledge of the high penalty jobs, no online (randomized) algorithm can achieve a competitive ratio of less than* $n^{\frac{1}{5}}$.

Proof. (Sketch) Consider the following scenario for a deterministic algorithm. The adversary gives two streams, each beginning at time $t = 0$.

Stream1 consists of k^2 jobs, each of size 1 and penalty k^2. Stream2 consists of k jobs each of size k and infinite penalty.
Depending on the remaining work of the online algorithm by time k^2, the adversary decides to give a third stream of jobs, or no more jobs. Stream 3 consists of $m = k^4$ jobs, each of size 1 and infinite penalty. Let y denote the total remaining work of jobs of Stream2 that are left at time $t = k^2$. The adversary gives Stream3 if $y \geq k^2/2$.
In either case, one can show that the ratio of the optimal cost to the online cost is at $\Omega(k)$, which implies a competitive ratio of $\Omega(n^{1/4})$. Due to lack of space, the details are deferred to the full version.
Clearly, the lower bound extends to the randomized case, as the adversary can simply send Stream3 with probability 1/2. Finally, to obtain a lower bound on competitive ratio in terms of C, we simply replace the infinite penalties of jobs in Stream2 and Stream3 by penalties of k^4.
The bound for the case when the high penalty jobs are known is similar and deferred to the full version of the paper.

4.2 Algorithm with Resource Augmentation

Now we will give a resource augmentation result for the weighted case with arbitrary penalties. The resource augmentation model is the one introduced Kalyanasundaram and Pruhs [13], where the online algorithm is provided a $(1+\epsilon)$ times faster processor than the optimum offline adversary.
Consider first, a fractional model where we can reject a fraction of a job. Rejecting a fraction f of job j has a penalty of fc_j. The contribution to the flow time is also fractional: If an f fraction of a job is remaining at time t, it contributes fw_j to the weighted flow time at that moment.
Given an instance of the original problem, create a new instance as follows: Replace a job j of size p_j, weight w_j and penalty c_j, with c_j jobs, each of weight w_j/c_j, size p_j/c_j and penalty 1.
Using the $O(\log^2 W)$ competitive algorithm for the case of arbitrary weights and uniform penalty, we can solve this fractional version of the original instance to within $O((\log W + \log C)^2)$. Now we use a $(1+\epsilon)$ speed processor to convert the fractional schedule back to a schedule for the original metric without too much blowup in cost, as described below.
Denote the fractional schedule output in the first step by S_F. The algorithm works as follows: If S_F rejects more than an $\epsilon/2$ fraction of some job, reject the job completely. Else, whenever S_F works on a job, work on the same job with a $(1+\epsilon)$ speed processor. Notice that when the faster processor finishes the job, S_F still has $1 - \epsilon/2 - 1/(1+\epsilon) = O(\epsilon)$ fraction of the job present.
We lose at most $2/\epsilon$ times more than S_F in rejection penalties, and at most $O(1/\epsilon)$ in accounting for flow time. Thus we have the following theorem:

Theorem 5. *The above algorithm is $O(\frac{1}{\epsilon}(\log W + \log C)^2)$-competitive for the problem of minimizing weighted flow time with arbitrary penalties on a $(1+\epsilon)$-speed processor.*

5 Conclusion

In this paper, we give online algorithms for the problems of minimizing flow time and job idle time when rejections are allowed at some penalty, and examine a number of problem variants. There are several problems left open by our work. It would be interesting to close the gap between the 1.5 lower bound and our 2-competitive algorithm for minimizing flow time with uniform penalties. The hardness of the offline version for the case of flow-time with uniform penalties is also not known[2].

References

1. Smith, W.: Various optimizers for single stage production. Naval Research Logistics Quarterly **3** (1956) 59–66
2. Chekuri, C., Khanna, S.: Approximation schemes for preemptive weighted flow time. ACM Symposium on Theory of Computing (STOC) (2002)
3. Chekuri, C., Khanna, S., Zhu, A.: Algorithms for weighted flow time. STOC (2001)
4. Bansal, N., Dhamdhere, K.: Minimizing weighted flow time. In: ACM-SIAM Symposium on Discrete Algorithms (SODA). (2003) 508–516
5. Becchetti, L., Leonardi, S., Spaccamela, A.M., Pruhs, K.: Online weighted flow time and deadline scheduling. In: RANDOM-APPROX. (2001) 36–47
6. Borodin, A., El-Yaniv, R.: On-Line Computation and Competitive Analysis. Cambridge University Press (1998)
7. Bartal, Y., Leonardi, S., Marchetti-Spaccamela, A., Sgall, J., Stougie, L.: Multiprocessor scheduling with rejection. In: ACM-SIAM Symposium on Discrete Algorithms (SODA). (1996)
8. Seiden, S.S.: Preemptive multiprocessor scheduling with rejection. Theoretical Computer Science **262** (2001) 437–458
9. Hoogeveen, H., Skutella, M., Woeginger, G.: Preemptive scheduling with rejection. European Symposium on Algorithms (2000)
10. Engels, D., Karger, D., Kolliopoulos, S., Sengupta, S., Uma, R., Wein, J.: Techniques for scheduling with rejection. European Symposium on Algorithms (1998) 490–501
11. Marshall, A.W., Olkin, I.: Inequalities: Theory of Majorization and Its Applications. Academic Press (1979)
12. Hardy, G., Littlewood, J.E., Polya, G.: Inequalities. Cambridge University Press (1952)
13. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. JACM **47** (2000) 617–643

[2] We can give a quasi-polynomial time approximation scheme (a $1+\epsilon$ approximation with running time $n^{O(\log n/\epsilon^2)}$). This is deferred to the full version of the paper.