Better Algorithms for Minimizing Average Flow-time on Related Machines

Naveen Garg Amit Kumar

Abstract

We consider the problem of minimising flow time on related machines and give an $O(\log P)$-approximation algorithm for the offline case and an $O(\log^2 P)$-competitive algorithm for the online version. This improves upon the previous best bound of $O(\log^2 P \log S)$ on the competitive ratio. Here P is the ratio of the maximum to the minimum processing time of a job and S is the ratio of the maximum to the minimum speed of a machine.

1 Introduction

A well-studied setting in the scheduling literature is one where we have multiple machines of differing speeds. This is commonly referred to as the related machine scenario. All machines are equally capable so that a job can be scheduled on any machine. The only distinction is that a machine of speed twice that of another machine would take half the time to finish the same job.

The jobs arrive over time and have to be scheduled so that the total flow time is minimized. The flow time of a job is the difference between its completion and release times and is equal to the total time that the job is waiting or being processed. We will permit preemption, so that a job can be stopped even before it is completed and resumed later. However, we will not permit migration, *ie.* the job cannot be resumed, after preemption, on another machine. Thus we are looking for a preemptive, non-migratory schedule on m related machines which minimizes the total flow time. In the three field notation of scheduling problems [5] this is denoted by $Q|r_j, pmtn| \sum_j C_j - r_j$.

When all machines have the same speed — the setting of parallel machines — the problem is well-studied. Leonardi and Raz [6] were the first to give an online algorithm for this problem with a bounded competitive ratio. They showed that the Shortest-Remaining-Processing-Time (SRPT) rule gives a schedule which is $O(\min(\log P, \log(n/m)))$-competitive, where P is the ratio of the maximum processing time to the minimum processing time and n is the number of jobs. They also established an $\Omega(\log P)$ lower bound on the competitiveness of any randomized online algorithm. The schedule obtained by Leonardi and Raz is migratory. Awerbuch et.al. [2] gave an online non-migratory algorithm with a competitive ratio of $O(\min(\log P, \log n)))$. This algorithm had the notion of a central queue where a job would wait till it was scheduled on one of the m machines. Avrahami and Azar [1] obtained an algorithm without a central queue so that now a job is dispatched to the appropriate machine as soon as it is released. Chekuri et.al.[3] presented a simple $O(\min(\log P, \log n/m)))$-competitive non-migratory algorithm that scheduled jobs based on their class instead of the remaining processing time.

For the case of related machines, the first algorithm with a bounded competitive ratio was obtained by the authors [4]; this algorithm had a $O(\log^2 P \log S)$ competitive ratio, where S is the ratio of the maximum machine-speed to the minimum speed.

In this paper we develop a new linear programming approach to minimizing flow time on related machines. Our LP, which is a time-indexed formulation is, seemingly, quite weak, since a fractional solution may schedule a job simultaneously on all machines. The flow time of a job is captured as the difference between its release time and the average time the job is scheduled in the LP solution and this quantity could be much smaller than the actual flow time of the job. In spite of all this we show a way of rounding this LP solution to obtain a non-migratory schedule whose flow time is at most $O(\log P)$ times the objective value of the LP solution. This gives an offline, $O(\log P)$-approximation algorithm for this problem.

We do not know how to solve this LP in an online manner. However, we can modify the linear program so that an optimum solution to the modified LP can be found in an online manner. Our procedure for rounding the LP solution into a non-migratory schedule continues to apply even in this online setting. Finally we show that the optimum solution to our modified LP is at most $O(\log P)$ times the flow time of the best schedule. This implies an $O(\log^2 P)$-competitive online algorithm for minimizing flow time on related machines.For the case of parallel machines, we do not need to modify the LP to solve it online and so we obtain an $O(\log P)$ competitive algorithm for parallel machines.

We remark that for both the offline and the online algorithm we obtain non-migratory schedules that are, respectively, within $O(\log P)$ and $O(\log^2 P)$ of the best offline migratory schedule. This also shows that permitting job-migration does not reduce flowtime by more than an $O(\log P)$ factor.

2 Preliminaries

We consider the scheduling problem where we are given machines with different speeds. For ease of notation, we shall work with *slowness* rather than speed. The slowness of a machine is defined as the reciprocal of its speed. In other words, a job j of size p_j will require $p_j \cdot s$ amount of processing time to get completely processed by a machine of slowness s.

There are m machines and let $s_i, s_i \geq 1$ be the slowness of machine i. The machines are ordered so that $s_1 \leq s_2 \leq \cdots \leq s_m$. There are n jobs. Job j has size p_j, and as mentioned above it takes $p_j \cdot s_i$ amount of time to finish on machine i. Let r_j be the release time of job j. We say that a job j is of *class* k if $2^{k-1} \leq p_j < 2^k$. Let the minimum size of a job be 1 and P be the maximum size of a job. We shall assume, without loss of generality, that all quantities are integers.

A schedule S specifies which job gets processed on each machine at each unit of time. Of course a job can start processing only after its release date. In this paper, all our schedules will be non-migratory and pre-emptive. However, our algorithms might construct migratory schedules on way to constructing non-migratory schedules. Also note that we can restrict ourselves to only those schedules which pre-empt jobs at integral time steps.

Let OPT denote the optimal schedule. We first consider the off-line problem where the sizes and the release dates of all the jobs are known at the beginning. Then we consider the on-line version of this problem where jobs arrive over time.

3 Off-line Algorithm

In this section we give a polynomial time $O(\log P)$-approximation algorithm for this problem. We first give an linear programming formulation for this problem and then show how to convert an optimum LP solution to a non-migratory schedule.

3.1 Linear Programming Formulation

We formulate the problem as an integer program. In this formulation, we will even allow migratory schedules as feasible solutions. In fact we will even allow the schedule to process the same job simultaneously over multiple machines. The lower bound generated from such a formulation will be enough for our purposes. For each job j, machine i and time t, we have a variable $x_{i,j,t}$ which is 1 if machine i processes job j from time t to $t+1$, 0 otherwise. The integer program is as follows. The variable j refers to jobs, i to machines and t to time.

$$\min \sum_j \sum_i \sum_t \quad x_{i,j,t} \cdot \left(\frac{t - r_j}{p_j \cdot s_i} + \frac{1}{2} \right) \tag{1}$$

$$\sum_j x_{i,j,t} \leq 1 \qquad \text{for all machines } i \text{ and time } t \tag{2}$$

$$\sum_i \sum_t \frac{x_{i,j,t}}{s_i} = p_j \qquad \text{for all jobs } j \tag{3}$$

$$x_{i,j,t} = 0 \qquad \text{if } t < r_j, \text{ for all jobs } j, \text{ machines } i, \text{ time } t \tag{4}$$

$$x_{i,j,t} \in \{0,1\} \qquad \text{for all jobs } j, \text{ machines } i, \text{ time } t \tag{5}$$

Constraint (2) refers to the fact that a machine can process at most one job at any point of time. Equation (3) says that job j gets completed by the schedule. Equation (4) denotes the simple fact that we cannot process a job before its release date. It should be clear that any integral solution gives rise to a schedule where jobs can be migrated across machines and may even get processed simultaneously on different machines. The only non-trivial equation in the integer program is the objective function. The following lemma shows that the optimal value of this integer program is a lower bound on the total flow-time of any non-migratory schedule. Let $\Delta_j(x)$ denote the term $\sum_i \sum_t x_{i,j,t} \cdot \left(\frac{t-r_j}{p_j \cdot s_i} + \frac{1}{2}\right)$. So the objective function is to minimize $\Delta(x) = \sum_j \Delta_j(x)$.

Let S be a non-migratory schedule. S also yields a solution to the integer program in a natural way – let x' denote this solution.

Lemma 3.1 *The total flow-time of S is at least $\sum_j \Delta_j(x')$.*

Proof: Fix a job j and suppose S schedules it on machine i. Let the completion time of j in S be t. So its flow-time is $t - r_j$. Notice that $\Delta_j(x')$ is maximized when j is scheduled from $t - p_j \cdot s_i$ to t. So

$$\Delta_j(x') \leq \sum_{t'=1}^{p_j \cdot s_i} \left(\frac{t - r_j - t'}{p_j \cdot s_i} + \frac{1}{2} \right) \leq t - r_j.$$

Thus $\sum_j \Delta_j(x')$ is at most the total flow time of S. ■

Recall that OPT is the optimal non-migratory schedule. The lemma above says that integer program yields a lower bound on the total flow-time of OPT. For the purpose of subsequent discussion, we modify the integer program slightly. We replace the objective function $\sum_j \Delta_j(x)$ by $\sum_j \Delta'_j(x)$, where $\Delta'_j(x) = \sum_i \sum_t x_{i,j,t} \left(\frac{t-r_j}{\lceil p_j \rceil \cdot s_i} + \frac{1}{2} \right)$. Here $\lceil p_j \rceil$ equals p_j rounded up to the nearest power of 2. Note that $\lceil p_j \rceil$ is the same for all jobs j of the same class. Thus the terms in this new objective function can be grouped together in a convenient manner. Since $\lceil p_j \rceil \geq p_j$, the optimal value of the integer program cannot increase and so the statement of Lemma 3.1 still holds. Further note that $\lceil p_j \rceil \leq 2 \cdot p_j$, and so the optimal value of the integer program does not change by more than a factor of 2. Let $\Delta'(x) = \sum_j \Delta'_j(x)$.

We now relax the integer program by replacing the constraints (5) by $0 \leq x_{i,j,t} \leq 1$. Let x^* be an optimal fractional solution this linear program.

3.2 Building the non-migratory schedule

We now show how to get a non-migratory schedule Q. Let y be the solution to the LP corresponding to the schedule Q. The solution y is related to the optimum solution x^* in the following manner: If slot (i,t) is full in x^*, *ie.*$\sum_j x^*_{i,j,t} = 1$, then we will have that $\sum_{j \in J_k} y_{i,j,t} \leq \sum_{j \in J_k} x^*_{i,j,t}$, where J_k denotes jobs of class k. However, if some part of slot (i,t) is empty in x^*, *ie.*$\sum_j x^*_{i,j,t} < 1$, then this empty part can be used in y to schedule jobs of any class.

For each slot (i,t) and class k we associate a variable $z_{i,k,t}$ which is initially equal to $\sum_{j \in J_k} x^*_{i,j,t}/s_i$ and denotes the total *volume* of class k jobs which have been processed in slot (i,t) in the solution x^*. In the course of our algorithm we reduce $z_{i,k,t}$ till it becomes zero.

We now describe our procedure for obtaining Q. We consider the slots in increasing order of time. At time t, consider machine i and class k. If there is a job, j, of class k which has been scheduled (in Q) on machine i and not completed yet, we schedule j in this slot to the *maximum possible extent.* This is the minimum of two quantities — the space available in this slot which is $\sum_{j \in J_k} x^*_{i,j,t}$ and the remaining processing of this job.

If there is no job, j as above, or if we finish the job in this slot, we need to identify another job of class k to schedule in this slot. If there is a job, j, of class k which has been released but not scheduled on any machine and if $p_j \leq \sum_{t' \leq t} \sum_{i'=i}^{m} z_{i',k,t'}$ then we schedule j in this slot to the maximum possible extent. Further, we reduce the variables $z_{i',k,t'}, i \leq i' \leq m, t' \leq t$ by a total amount equal to p_j. The variables are reduced in increasing order of (i',t').

For each slot (i,t) the above procedure is repeated for every class k. This ensures that for every class k, $\sum_{j \in J_k} y_{i,j,t} \leq \sum_{j \in J_k} x^*_{i,j,t}$. If some part of this slot is empty in x^* we try to use it in solution y by using the above procedure to find a job (of any class) which can be scheduled in this slot. This job is then assigned to the maximum possible extent. The process may have to be repeated till this slot is fully occupied.

We must show that this procedure will eventually schedule all the jobs. Suppose for the sake of contradiction there is a job j which does not get assigned to a machine in our schedule. Let k denote its class. Let t be the time by which all jobs finish processing in schedule Q. Initially, the sum $\sum_i \sum_t z_{i,k,t}$ equals the total processing time of all jobs in class k and every time we schedule a job of class k we remove an amount equal to its processing time from this sum. This implies that eventually, this sum is exactly p_j and so j will get scheduled when we encounter an empty slot at time t. This yields a contradiction. Thus, our algorithm yields a feasible non-migratory schedule.

Let $P(x^*)$ denote the total time for which jobs are processed in solution x^*, i.e., $P(x^*) = \sum_i \sum_j \sum_t x^*_{i,j,t}$. We define $P(y)$ similarly.

Lemma 3.2 *$P(y) \leq P(x^*)$.*

Proof: When we assign a job of class k to machine i in schedule Q, we reduce $\sum_{i'=i}^{m} \sum_{t' \leq t} z_{i',k,t'}$ by a total amount p_j. Since $s_{i'} \geq s_i$, the reduction in $\sum_{i'=i}^{m} \sum_{t' \leq t} s_{i'} \cdot z_{i',k,t'}$ is greater than $s_i * p_j$, the processing time of j in Q. Note that the sum $\sum_i \sum_t s_i \cdot z_{i,k,t}$ equals the total processing time of all class k jobs in the solution x^*. This implies that the total processing time of class k jobs is less in y than in x^*. ■

Now we try to bound the flow-time of our schedule.

Lemma 3.3

$$\Delta'(y) \leq \Delta'(x^*) + 2 \log P \cdot P(x^*).$$

Proof: Note that

$$\Delta'(y) - \Delta'(x^*) = \frac{P(y) - P(x^*)}{2} + \sum_i \sum_j \sum_t \frac{(y_{i,j,t} - x^*_{i,j,t}) \cdot t}{s_i \cdot \lceil p_j \rceil} \leq \sum_k \sum_i \sum_{j \in J_k} \sum_t \frac{(y_{i,j,t} - x^*_{i,j,t}) \cdot t}{s_i \cdot 2^k}.$$

where the last inequality follows from Lemma 3.2.

For a fixed class k and machine i, consider the quantity $\sum_t \sum_{j \in J_k} (y_{i,j,t} - x^*_{i,j,t}) * t / s_i$. A little thought yields that this is same as the sum over all t of the difference in volume of jobs of class k which get processed on machine i after t. Summing it over all i, this is same as the sum over t of the difference in the volume of jobs of class k which get processed after t in the two schedules.

Define $V_{k,t}(x^*)$ as the volume of class k jobs which get processed by time t in x^*. Define $V_{k,t}(y)$ similarly. Since the total volume of jobs of a particular class that get processed in the two schedules is the same, we see that for any fixed k,

$$\sum_i \sum_t \sum_{j \in J_k} \frac{(y_{i,j,t} - x*_{i,j,t}) \cdot t}{s_i \cdot 2^k} = \sum_t (V_{k,t}(x^*) - V_{k,t}(y))/2^k$$

.

Claim 3.1 *$V_{k,t}(x^*) - V_{k,t}(y)$ is at most $m_t \cdot 2^{k+1}$, where m_t is the number of busy machines in our schedule Q at time t.*

Proof: Consider the quantity $\sum_i \sum_{t' \leq t} z_{i,k,t'}$. Initially this is equal to $V_{k,t}(x^*)$. Every time we schedule a job of class k we reduce this quantity by the processing time of the job. Hence by time t, the reduction in this quantity equals the total volume of jobs that have been scheduled in Q. Let $z^t_{i,k,t'}$ denote the value of the variable $z_{i,k,t'}$ after our algorithm has processed slot (i,t). Thus $V_{k,t}(x^*) - \sum_i \sum_{t' \leq t} z^t_{i,k,t'}$

equals $V_{k,t}(y)$ plus "the unprocessed volume of class k jobs that have been scheduled but have not finished by time t in Q".

To bound $\sum_i \sum_{t' \le t} z^t_{i,k,t'}$, we fix a machine i and consider the sum $\sum_{t' \le a} z^a_{i,k,t'} + p^a$, where p^a is the unprocessed volume of the class k job that is scheduled but has not finished at time a. We argue that for any a this sum can never exceed 2^k.

When machine i has an unfinished job of class k, the increase $\sum_{t' \le a+1} z^{a+1}_{i,k,t'} - \sum_{t' \le a} z^a_{i,k,t'}$ is offset by the decrease $p^a - p^{a+1}$. Thus, till i has an unfinished job, this sum cannot increase. If we schedule a new job from slot $(i, a+1)$, then p^{a+1} is balanced by the decrease $\sum_{t' \le a+1} z^{a+1}_{i,k,t'} - \sum_{t' \le a} z^a_{i,k,t'}$. This is because, when we scheduled j we reduced $z_{i,k,t'}, t' \le a+1$ by an extent of p_j.

Thus, this sum can increase beyond 2^k only if no job of class k is scheduled in slot $(i, a+1)$. If $\sum_{t' \le a+1} z^{a+1}_{i,k,t'} > 2^k$ then the total volume of class k jobs processed in x^* by time $a+1$ exceeds the total volume of class k jobs scheduled in y till time $a+1$ by at least 2^k. This implies that there is a job, j, of class k which has been released but not scheduled in Q till time $a+1$. Since $p_j \le 2^k$, our procedure would have scheduled j at slot $(i, a+1)$ and this leads to a contradiction.

Now suppose there are idle machines at time t in our schedule Q. Let i be such a machine of smallest index. Then we claim that $\sum_{i'=i}^{m} \sum_{t' \le t} z^t_{i',k,t'}$ is at most 2^k. Indeed if it were greater than 2^k, then, as above, we can argue that there must be a job j of class k which has been released by time t but not scheduled in Q. Then machine i should not be idle at this time.

Combining these two arguments, we see that $\sum_i \sum_{t' \le t} z^t_{i,k,t'}$ plus "the total unprocessed volume of class k jobs which have been scheduled but not finished by time t" is at most $m_t \cdot 2^k + 2^k$. Thus $V_{k,t}(x^*) - V_{k,t}(y)$ is at most $(m_t + 1)2^k$ which, since $m_t \ge 1$ is at most $m_t \cdot 2^{k+1}$. ■

Thus we get

$$\Delta'(y) - \Delta'(x^*) \le \sum_k \sum_t m_t \cdot 2^{k+1}/2^k = (2 \cdot \log P) \sum_t m_t = 2 \log P \cdot P(y) \le 2 \log P \cdot P(x^*),$$

where the last inequality follows from Lemma 3.2. ■

The solution y can be converted into a schedule Q by splitting each slot (i, t) into sub-slots of length proportional to $y_{i,j,t}$ and scheduling job j in the sub-slot corresponding to it.

We are ready to relate the flow-time of the schedule Q to that of OPT.

Lemma 3.4 *The total flow-time of our schedule Q is at most $2\Delta'(y) + P(y) \cdot \log P$.*

Proof: Fix a job j. Let C_j be its completion time in schedule Q. It is easy to see that $2\Delta'_j(y)$ is at least the flow-time of j minus "the total time for which j is preempted in schedule Q". Since on any machine, there can be at most one job of class k which is preempted, we see that the sum $2\sum_{j \in J_k} \Delta'_j(y)$ is at least the total flow time of all class k jobs minus "the total processing time of schedule Q". Thus we get that the flow-time of our schedule is at most $2\Delta'(y) + \log P \cdot P(y)$. ■

We are now ready to state the main theorem.

Theorem 3.5 *There is a polynomial time $O(\log P)$ approximation algorithm for minimizing flow time on related machines.*

Proof: Combining Lemmas 3.4, 3.2 and 3.3, we see that the flow-time of schedule Q is at most $5P(x^*)\log P + 2\Delta'(x^*)$. Now $P(x^*)$ is at most twice of $\Delta'(x^*)$ (because of the additive 1/2 term in the objective function). Lemma 3.1 finally implies the theorem. ■

4 The On-line algorithm

We now give an $O(\log^2 P)$-competitive algorithm for this problem. It is easy to see that a solution to the linear program can be rounded into a non-migratory solution in an on-line manner. However, we do not know how to find an optimum solution to the linear program in an online manner. In this section we first present a modified linear program and argue that its optimum solution can be computed online. We then relate the optimum of this linear program to the flow-time of the optimal schedule.

Let x be a solution to the linear program. We now define the fractional flow time of job j, $\Delta''_j(x)$ as

$$\Delta''_j(x) = \sum_i \sum_t x_{i,j,t} \cdot \frac{t - r_j}{\lceil p_j \rceil \cdot s_i}.$$

Note that this is only less than the fractional flow-time, as defined earlier and so the sum of $\Delta''_j(x)$, which is the objective function, continues to be a lower bound on the optimum.

With this modification we cannot claim any more that $P(x) \le 2\sum_j \Delta''_j(x)$. Bounding the processing time of the fractional solution was crucial to the analysis of the off-line algorithm. Our second modification to the linear program lets us do this. We now require that a job j is scheduled on machine i at least $\lceil p_j \rceil \cdot s_i$ time units after its release. We encode this in the linear program by changing constraint (4) to $x_{i,j,t} = 0$ if $t \le r_j + \lceil p_j \rceil \cdot s_i$. However, now it is no more true that the objective function is a lower bound on the optimum flow time.

The objective function of this modified linear program is equivalent to minimizing $\sum_j \sum_i \sum_t \frac{x_{i,j,t} \cdot t}{\lceil p_j \rceil \cdot s_i}$. This is because of the constraint (3) in the linear program. We claim that the following greedy procedure yields an optimum solution to the linear program : at each time t consider the machines in increasing order of slowness. On machine i schedule the job, j, of smallest class which is available ($t \ge r_j + s_i \cdot \lceil p_j \rceil$) and not completed ($\sum_t \sum_i x_{i,j,t}/s_i < p_j$). Schedule this job to the maximum extent possible (i.e., until the slot is fully occupied or this job finishes) and continue. Note that $x_{i,j,t}$ denotes the fraction of time during $(t, t+1)$ for which j was scheduled on i.

We now argue that the solution obtained by this greedy procedure is optimal.

Lemma 4.1 *The solution x obtained by the greedy algorithm is an optimal solution to the linear program.*

Proof: Let O' be an optimal solution to the modified LP and let x^* denote the solution corresponding to O'. For a class k of jobs, machine i and time t, define $u^*_{i,k,t}$ as $\sum_{j \in J_k} x^*_{i,k,t}$, i.e., the total processing done on class k during $(t, t+1)$ on i. Define $u_{i,k,t}$ with respect to the greedy solution x similarly. Clearly it is enough to show that $u_{i,k,t} = u^*_{i,k,t}$ for all i, k, t.

So let us suppose for the sake of contradiction that greedy is not optimal. Suppose t is the first time at which the two solution differ in the class of job scheduled, i.e., there is a class k and machine i

for which $u^*_{i,k,t} \neq u_{i,k,t}$ (if there are multiple machines i at time t for which this happens, then pick the machine with the smallest index). Then there are jobs j, j' of different class such that greedy processed more of j than j' during $(t, t+1)$ on i while the converse is true for O'. So O' must have processed j at some other time $t' \geq t$ on some other machine i' (the pair (i,t) and (i',t') are different). We reduce $x^*_{i,j',t}$ by an amount ϵ. Then ϵ/s_i amount of job j' has to be accommodated at the slot on machine i' at time t'. So we increase $x^*_{i',j',t'}$ by $\epsilon \cdot s_{i'}/s_i$. To avoid violating the constraint (2) at i', t', we reduce $x^*_{i',j,t'}$ by $\epsilon \cdot s_{i'}/s_i$ and increase $x^*_{i,j,t}$ by ϵ. Clearly we have a valid solution. Let us see the change in the objective function for x^*. The change is equal to

$$\frac{t}{s_i} \cdot \left(\frac{\epsilon}{\lceil p_j \rceil} - \frac{\epsilon}{\lceil p_{j'} \rceil} \right) + \frac{t'}{s_{i'}} \cdot \left(\frac{\epsilon \cdot s_{i'}}{s_i \cdot \lceil p_{j'} \rceil} - \frac{\epsilon \cdot s_{i'}}{s_i \cdot \lceil p_j \rceil} \right) = (t - t') \frac{\epsilon}{s_i} \cdot \left(\frac{1}{\lceil p_j \rceil} - \frac{1}{\lceil p_{j'} \rceil} \right)$$

Since $\lceil p_j \rceil < \lceil p_{j'} \rceil$ and $t \leq t'$ the above quantity is non-positive. Thus we have brought O' closer to our schedule without increasing the cost. By repeatedly doing such operations we can make O' identical to the greedy schedule. ■

As in the offline case, we convert the greedy solution, x, to obtain a solution y which corresponds to a non-migratory schedule, Q. Define $\Delta''(y) = \sum_j \Delta''_j(y)$ for any feasible solution y. The following claim bounds the processing time of the greedy solution.

Claim 4.1 *For any feasible solution y, the total processing time, $P(y)$ is at most $\Delta''(y)$.*

Proof: $\Delta''(y) = \sum_{j,i,t} \frac{y_{i,j,t}(t-r_j)}{\lceil p_j \rceil \cdot s_i} \geq \sum_{j,i,t} y_{i,j,t} = P(y)$. The last inequality follows from the fact that $y_{i,j,t} > 0$ only if $t \geq r_j + \lceil p_j \rceil \cdot s_i$. ■

Claim 4.2 *The total flow time of schedule Q is at most $(4 + 9 \log P) \cdot \Delta''(x)$.*

Proof: It is easy to show that if a job gets released at r_j and finishes at t on some machine, its flow-time is at most $4 \cdot \Delta''_j(y)$ plus "the total time for which j was interrupted". Since the interruption intervals for jobs of the same class on a particular machine are disjoint, we can bound the total interruption time for all the jobs by $P(y) \cdot \log P$. So we get that the total flow time is at most $4 \cdot \Delta''(y) + P(y) \log P$. From Lemma 3.3, we know that $\Delta''(y)$ is at most $\Delta''(x) + 2 \cdot P(y) \cdot \log P$. This implies that the flow time of Q is at most $4 \cdot \Delta''(x) + 9 \cdot P(y) \cdot \log P$.

As in the offline case, we can argue that $P(y) \leq P(x)$. Combining this with the claim above completes the proof. ■

Finally we bound $\Delta''(x)$ in terms of the flow-time of the optimum schedule OPT (recall that OPT can schedule a job anytime after its release).

Claim 4.3 *$F(x)$ is at most $O(\log P)$ times the flow-time of* OPT.

Proof: Note that $F(x)$ is less than the optimum flow time of the schedule with the restriction that a job, j is scheduled on machine i at least $\lceil p_j \rceil \cdot s_i$ units after its release.

Now consider OPT. Fix a class k and let us only consider the time slots in OPT where we schedule class k jobs. Now we shift each class k job on a machine i by $2^k \cdot s_i$ units (where the shifting is done with respect to these slots only). Note that this shifting does not put any of these jobs into a slot of another class. The shifting process may occupy empty slots at the end of the schedule. If we do this for each k (starting from $k = 1$), we get a schedule which obeys the property that a job j, of class k, is scheduled on a machine i only after $r_j + 2^k \cdot s_i$ time.

Now note that the total increase in the flow time of jobs of class k is at most twice the processing time of the optimum schedule. This implies that the new schedule obtained has flow time at most $(1 + 2 \log P)$ times the flow-time of OPT. ■

Combining the last two claims we obtain the following theorem.

Theorem 4.2 *There is an online algorithm for minimizing flow time on related machines which has a competitive ration of $O(log^2 P)$ where P is the ratio of the maximum to the minimum processing time of a job.*

5 Open Problems

We believe that such a linear programming approach should lead to an offline approximation algorithm for minimizing flow time on unrelated machines. We leave this as an open problem. It would also be interesting to close the gap between the $O(\log^2 P)$ upper bound and the $\Omega(\log P)$ lower bound on the competitive ratio for flow time on related machines.

References

[1] Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *SPAA*, pages 11–18, 2003.

[2] Awerbuch, Azar, Leonardi, and Regev. Minimizing the flow time without migration. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1999.

[3] Chekuri, Khanna, and Zhu. Algorithms for minimizing weighted flow time. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2001.

[4] Garg and Kumar. Minimizing average flow time on related machines. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2006.

[5] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Mathematics*, 5:287–326, 1979.

[6] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. In *STOC*, pages 110–119, 1997.