Minimizing average flow-time : the unrelated case

Naveen Garg, Amit Kumar, and V.N. Muralidhara

Dept. of Computer Science and Engg.
IIT Delhi, New Delhi – 110016
{naveen,amitk,murali}@cse.iitd.ac.in

Abstract. We consider the problem of minimizing the flow-time in the unrelated machines setting.We give general techniques for getting good approximation algorithms for a large sub-class of this problem. As special cases, we get following results : (1) $O(K)$-approximation algorithm if the set of processing times of all the jobs (on all machines) is of size K, (2) $O(\log P)$-approximation algorithm if each job can only go on a specified subset of machines, but has the same processing requirement on each such machine. Further, the machines can have different speeds. Here P is the ratio of the largest to the smallest processing requirement, (3) constant approximation algorithm for the unrelated case if we assume that our algorithm has slightly more resources than the optimum. Another merit of our algorithm is that it is very simple, and considerably simplifies known results for several special cases of the scheduling problem.

1 Introduction

We consider the problem of scheduling jobs on multiple machines so as to minimize the total flow time. The flow time of a job is the total time it spends in the system and equals the difference between its completion time and release time. A job j has a processing time p_{ij} on machine i; this is the most general model of processing times and is known as the *unrelated machines model.* A special case is the *related machines model* where job j has length a_j, machine i has speed (or slowness) s_i and the processing time of job j on machine i is $a_j \cdot s_i$. When the machines are identical so that $s_i = 1$ for all i, then we have the *parallel machines model.*

Minimizing flow time is most interesting in the online setting where the processing time of a job on various machines is revealed only when the job is released. For parallel machines an $O(\log P)$-competitive algorithm was first given by Leonardi and Raz [5] while for related machines an $O(\log^2 P)$ algorithm was given by Garg and Kumar [3], where P is the ratio of the maximum and minimum processing times. However, for unrelated machines no online algorithm can have bounded competitive ratio and this is true even for the very restricted setting when each p_{ij} is either 1 or ∞[4]. It is for this reason that we consider offline algorithms for minimizing flow time on unrelated machines.

In this paper we present a new algorithm for this problem. The setting when the processing times p_{ij} are completely arbitrary seems very unnatural and difficult to approximate. In this paper we consider a model which tries to capture the variation in processing times which we motivate as follows. Suppose we have a cluster of heterogeneous computers (machines) on which we want to run some programs (jobs). These programs are solving different problems – matrix multiplication, sorting, sequence alignment, mpeg decoding etc. Similarly machines have different architectures – super scalars, multi-cores, vector machines – and some architectures might be better suited for a particular problem. This naturally gives rise to different processing times for different problems on different architectures; we assume that these processing times are from a set $B = \{b_1, b_2 \ldots, b_\beta\}$. Two machines which implement the same architecture could run at different speeds due to differences in clock frequency, memory etc. This results in processing time variations which would (to a large extent) be independent of the program being run. Similarly, we could be using different algorithms to solve the same problem and running them on different instances. The variation in the processing times of the resulting programs would therefore be independent of the machine on which the program is executed.

We say that processing times have an (α, β)-*variability* if the processing time of job j on machine i can be expressed as $p_{ij} = a_j \cdot b_{ij} \cdot s_i$ where $b_{ij} \in B, |B| = \beta$ and $1 \le a_j \le \alpha$. Our main result in this paper is an $O(\beta(1 + \log \alpha))$-approximation when processing times have an (α, β) variability. Note that when p_{ij} take k different values then we can always set $\alpha = 1, \beta = k$ and this gives an $O(k)$ approximation. This result was obtained independently by Sitters [6] using a different algorithm. However, our aim in this paper is to do even better by providing an algorithm which can exploit bounded variability in the processing times. As a case in point, consider the setting when job j has processing time p_j but it can be scheduled only on a subset of machines S_j. If all machines have the same speed, we call this the subset-parallel model. Garg and Kumar [4] gave an $O(\log P)$ approximation for minimizing flow time in this model, where $P = \max_j p_j / \min_j p_j$. If machine i has speed s_i then the processing time of job j on machine i (assuming $i \in S_j$) is $p_{ij} = p_j \cdot s_i$. Note that choosing $B = \{1, \infty\}$ implies a $(P, 2)$ variability on the processing times and hence our algorithm gives an $O(\log P)$-approximation. This is the first approximation algorithm for this *subset-related model*. We believe that for many other natural settings, the variations in processing times can be captured using (α, β) variability for small α, β and this would then imply better approximation guarantees using our algorithm.

The difficulty in approximating flow time for unrelated machines led us to consider the problem with resource augmentation. In this setting the approximation algorithm has machines which are an ε-factor faster than those of opt. This means that the approximation algorithm can complete $1 + \varepsilon$ units of work in 1 time unit while opt can only do 1 unit. Such a model was considered for parallel machines and Chekuri et. al. [1] showed a $O(\varepsilon^{-4})$-competitive algorithm for minimizing flow timex. A small modification to our main algorithm gives an $O(\varepsilon^{-1} \log \varepsilon^{-1})$-approximation for minimizing flow time on unrelated machines. Note that this result does not assume that processing times have bounded variability.

Our final result in this paper improves the lower bound on the approximation ratio for minimizing flow time on parallel machines from $\Omega(\sqrt{\log P/\log\log P})$ [4] to $\Omega(\log^{1-\varepsilon} P)$ for any $\varepsilon > 0$.

2 Unrelated machines

We consider the problem of minimizing average flow-time of n jobs on m machines. Job j has a release date r_j. If j is scheduled on machine i then it has to be processed for p_{ij} time units. We allow for preemption but disallow migration, i.e. a job cannot be interrupted on a machine and restarted on another machine. Let $\mathcal{S}$ be a scheduling algorithm. The completion time of a job j in $\mathcal{S}$ is denoted by $C_j^{\mathcal{S}}$. The flow time of j in $\mathcal{S}$ is defined as $F_j^{\mathcal{S}} = C_j^{\mathcal{S}} - r_j$. Our goal is to find a scheduling algorithm which minimizes the total flow time of jobs.

We now give an integer programming formulation for minimizing flow time on unrelated machines [3]. 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.

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

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

$$\sum_i \sum_t x_{i,j,t}/p_{ij} = 1 \qquad \text{for all jobs } j \tag{3}$$

$$x_{i,j,t} = 0 \qquad \text{if } t < r_j \tag{4}$$

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

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) captures the requirement that we cannot process a job before its release date or schedule a job j on a machine not in the set $S(j)$. It should be clear that an integral solution gives rise to a schedule where jobs can migrate across machines and may even be processed simultaneously on different machines.

The only non-trivial part of the integer program is the objective function. Let $\Delta_j(x) = \sum_i \sum_t x_{i,j,t} \cdot \left(\frac{t-r_j}{p_j} + \frac{1}{2}\right)$. So the objective function is to minimize $\Delta(x) = \sum_j \Delta_j(x)$. Let $\mathcal{S}$ be a non-migratory schedule. $\mathcal{S}$ also yields a solution to the integer program in a natural way — let $x^{\mathcal{S}}$ denote this solution. The following fact was established in [3].

Fact 21 *The total flow-time of $\mathcal{S}$ is at least $\sum_j \Delta_j(x^{\mathcal{S}})$.*

We say that the processing times have variability (α, β) if for all machines i and job j, $p_{ij} = a_j \cdot b_k \cdot s_i$ where $1 \leq a_j \leq \alpha$ and $b_k \in B$, where $B = \{b_1, \ldots, b_{|B|}$ is a set of size at most β. We impose no restriction on s_i but it is no loss of generality to assume that $s_i \geq 1$.
We say that job j on machine i is of *class* k if $p_{ij} = a_j \cdot b_k \cdot s_i$. Consider the integer program (1). For the purpose of subsequent discussion, we modify the integer program slightly by replacing the objective function $\sum_j \Delta_j(x)$ by $\sum_j \Delta'_j(x)$, where $\Delta'_j(x) = \sum_i \sum_t x_{i,j,t} \left(\frac{t-r_j}{p'_{ij}} + \frac{1}{2} \right)$. Here $p'_{ij} = p_{ij}/a_j$. It is easy to see that this modification does not decrease the cost of a solution x and increases it by at most a factor α. But this slight modification in the objective function turns out to be very useful. Consider the slots in x for a fixed machine i and class k – if we rearrange the jobs being processed on these slots (without violating release dates), the objective function does not change.
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 to this linear program (LP).

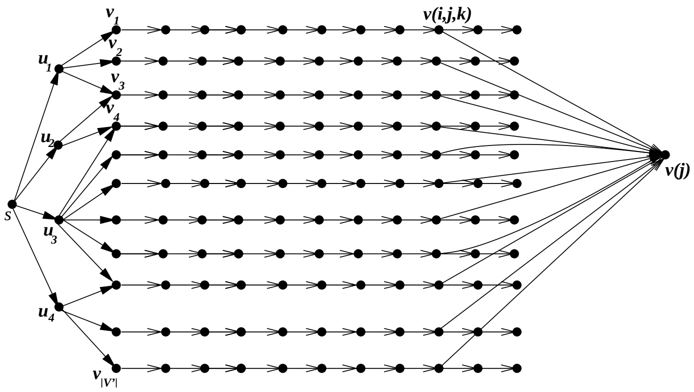
2.1 The Algorithm

We order the jobs in ascending order of their release times (breaking ties arbitrarily). We begin by constructing a directed graph G over which we would be running the unsplittable flow algorithm of Dinitz et.al. We first describe the vertex set V of G. For each machine i, job j and class k, we have a vertex $v(i, j, k)$. For each job j, we have a vertex $v(j)$ and a special source vertex s. Further we have a set of special vertices $U = \{u_1, \ldots\}$ – the size of U will be the processing time of the optimal solution divided by α (this will become clear in the construction below). We now describe the edges in G. Since we shall treat this as an instance of the unsplittable flow problem, we need to define flow values on each edge and the demand of each vertex. We describe this next :

1. For each j, $1 \leq j \leq n-1$, there is an edge from $v(i, j, k)$ to $v(i, j+1, k)$ for every i, k. So for fixed values of i and k, we can think of this as a directed path joining the corresponding vertices in the order of release dates.
2. Let $z^*_{i,j} = \sum_t \frac{x^*_{i,j,t}}{p'_{ij}}$. This is the total (fractional) processing of j on machine i scaled by the factor a_j. If this quantity is non-zero, and the class of j on machine i is k, we have an edge from $v(i, j, k)$ to $v(j)$ with flow $z^*_{i,j}$. There are no other edges incident at vertex $v(j)$. Note that for each job j, the LP has a constraint $\sum_i \sum_t x_{i,j,t}/p_{ij} = 1$. This implies that the total flow reaching vertex $v(j)$ is exactly a_j and this is the demand of vertex $v(j)$. The demand of all other vertices in G is zero.
3. The edge from $v(i, j-1, k)$ to $v(i, j, k)$ has flow given by conservation constraints.
4. We now build a bipartite subgraph between the vertices in U and the set $V' = \{v(i, 1, k) : \text{for all } i, k\}$. We order the vertices in V' by descending value of $b_k \cdot s_i$ and relabel vertices as $v_1, v_2, \ldots, v_{|V'|}$ to reflect this ordering. Let f_i be the total flow leaving vertices $v_1, \ldots, v_i$.

With each vertex v_i we associate an interval $[f_{i-1}, f_i]$ (let $f_0 = 0$). Further for each vertex $u_l \in U$ we associate the interval $[\alpha\cdot(l-1), \alpha\cdot l]$. There is an edge from u_l to v_i iff the corresponding intervals overlap and the flow on this edge is the length of the overlap. Hence the total flow leaving u_l is exactly α (except perhaps for the last vertex in U) and the total flow entering v_i is the total flow leaving it.

5. For every $u_l \in U$, there is an edge from s to u_l and the flow on these edges is given by flow conservation.



Let f be the initial flow as described above. We will compute a flow, f^*, in which all demand from the source s to sink $v(j)$ is routed on a single path. The unsplittable flow theorem of Dinitz et.al. says that this can be done such that the flow on any edge exceeds the fractional flow on that edge by at most the maximum demand, which in our case is α. The unsplittable flow f^* in graph G, assigns each job to a unique machine by virtue of the fact that the flow from s to $v(j)$ is routed on a single path (and hence must pass through exactly one of the directed paths constructed in Step 1 above). Thus if the edge $(v(i, j, k), v(j))$ has non-zero flow in f^* then we schedule job j on machine i. Let P be the total processing time corresponding to this assignment of jobs to machines.
To do the actual assignment of jobs to slots on machine i, we consider jobs by increasing release time and schedule them in the earliest available slots (of class k on machine i). Note that we might not be able to assign all jobs to slots since we might run out of slots. It might also happen that only a part of a job can be assigned.

Claim. For each machine i and class k, at most α jobs will remain unassigned in the above step.

Proof. Fix a machine i and class k. The flow through the edge $(v(i, j - 1, k), v(i, j, k))$ in f^* exceeds the original flow through this edge by at most α. Since the total available space after t equals the original flow through this edge, the amount of additional available space needed to schedule jobs released at or after t is at most α. Since at most α jobs could have used this space, the number of unassigned jobs is at most α. ■

For now, we ignore the unassigned jobs and let J' be the jobs that could be assigned completely to slots above. Let x_1 be a solution to the LP that corresponds the above reassignment of jobs in J'.

Claim. $\sum_{j \in J'} \Delta'_j(x_1) \leq \sum_{j \in J} \Delta'_j(x^*)$.

Proof. Since we have only assigned class k jobs to the available space the fractional cost could not have increased. ■

For each machine i we consider all jobs in J' that are assigned to it and schedule them using the lowest-class-first rule, i.e., always give preference to jobs of lower class. Let S be this schedule and x_2 be the LP solution corresponding to it. The following claim was proved in [3]

Claim. $\sum_{j \in J'} \Delta'_j(x_2) \leq \sum_{j \in J'} \Delta'_j(x_1)$.

Schedule S has the nice property that a job of class k can only be interrupted by jobs of strictly lower class. This allows us to place a bound on the total flow time of jobs in S (the proof again follows from [3]).

Claim. Total flow time of jobs in S is at most $\sum_{j \in J'} \Delta'_j(x_2) + \beta P$.

We now add the jobs in $J - J'$ to schedule S by considering these jobs in an arbitrary order and scheduling them at the earliest possible time on the machine they were assigned to. Each of these jobs would have a flow time which is at most the total processing time on the machine. Since each machine can have at most $\alpha\beta$ of these jobs, the total increase in the flow time of schedule S due to these jobs is at most $\alpha\beta P$. Let S' be the new schedule and this would also be our final schedule. Combining the previous claims we get

Lemma 1. *Total flow time of jobs in S' is at most $\Delta'(x^*) + (\alpha + 1)\beta P$.*

Recall that P is the total processing time of the jobs in our schedule S'. The following lemma bounds P.

Lemma 2. *$P \leq 4 \cdot \Delta'(x^*) + 2\alpha \cdot T$ where T is the largest processing time of a job in S'.*

Proof. Let $f(i,k)$ be the total flow leaving vertex $v(i,1,k)$ in the flow f. It is easy to see that $\sum_{i,j,t} x^*_{i,j,t} = \sum_{i,k} s_i \cdot b_k \cdot f(i,k)$. Similarly, $P = \sum_{i,k} s_i \cdot b_k \cdot f'(i,k)$ where f' is the unsplittable flow we compute.

Claim. $\sum_{i,k} s_i \cdot b_k \cdot f'(i,k) \leq 2 \sum_{i,k} s_i \cdot b_k \cdot f(i,k) + 2\alpha T$

Proof. We shall split the sum in the left hand side as $\sum_{u_l \in U} F'(u_l)$, where $F'(u_l)$ is the portion of $\sum_{i,k} s_i \cdot b_k \cdot f'(i,k)$ where we only look at the flow $f'(i,k)$ which comes from the vertex u_l. Define $F(u_l)$ for the right hand side similarly.
First observe that the total flow leaving u_j is α in f and hence it is at most 2α in f'. Hence $F'(u_l) \leq 2F(u_{l-1})$, $l \geq 2$ – here we use the fact

that if flow goes from u_l to a vertex $v(i,1,k)$ and from u_{l-1} to $v(i',1,k')$ then $b_{k'}s_{i'} \geq b_k s_i$. All that remains is to bound $F(u_1)$. But this is at most $2\alpha T$. ■

Note that $\sum_{i,j,t} x^*_{i,j,t} \leq 2\Delta'(x^*)$. This together with the above claim implies the lemma. ■

We guess the longest processing time of any job (this can take only mn different values) in the optimum schedule, say L. We run the above algorithm with all $p_{ij} > L$ set to infinity. This ensures that $T \leq \mathsf{opt}$. Hence $P \leq (4+2\alpha)\mathsf{opt}$ and so the flow time of schedule S' is $\mathrm{O}(\alpha^2\beta\mathsf{opt})$.

Theorem 1. *The flow time of schedule S' is $O(\alpha^2\beta \cdot \mathsf{opt})$.*

3 Speed augmentation in unrelated machines

We now consider the setting where our algorithm has machines which are faster than the machines available to the optimum algorithm by a factor $(1+\varepsilon)$. In particular, this means that in each unit of time our algorithm can do $1+\varepsilon$ units of work.
We shall increase the speed in three phases – in each speed we shall assume that we can do an extra of $\varepsilon/3$ units of work in each unit time slot. We begin by rounding down all processing times to a power of $(1+\varepsilon/3)$. Note that the optimum solution to this instance has flow time only lower than the optimum flow time for the original instance. A solution to this rounded down instance also gives us a solution to the original instance since we can use an additional speed of $\varepsilon/3$ to process each job fully – this is the first phase where we give us extra speed.
We can run the algorithm from the previous section with $a_j = 1$ for all j. This would imply $\alpha = 1$ and $\beta = \log_{1+\varepsilon/3} P$ where $P = \max_{i,j} p_{ij}$ is the maximum processing time. This in turn implies an approximation guarantee of $\log_{1+\varepsilon/3} P$. However, in this section we present an $\mathrm{O}(1/\varepsilon)$-approximation algorithm. Note that a job is of class k on machine i, if its processing time on this machine is $(1+\varepsilon/3)^k$.
We first show an easier $O(1/\varepsilon^2)$-approximation algorithm. Our algorithm proceeds as the algorithm in the previous section (with $a_j = 1$). We build the same graph G and compute an unsplittable flow f' – note that while computing the flow, we do not assume machines have extra speed. This unsplittable flow is used to determine the assignment of jobs to machines as in the previous section. We also compute the final schedule S as before – recall that S schedules the jobs in J' only. Consider the p_{ij} slots in which job j is scheduled on machine i. We exploit the fact that we have an additional speed $\varepsilon/3$ and move the processing of job j to the first $\frac{p_{ij}}{(1+\varepsilon/3)}$ of these p_{ij} slots – this is the second phase where we give us extra speed. Let this schedule be S'. The following analogue of Claim 1 uses the fact that we can get an improved result because of the extra speed.

Claim. The total flow time of jobs in S' is at most $O\left(\frac{1}{\varepsilon}\right) \cdot \sum_{j \in J'} \Delta'_j(x_2)$.

For each machine i we have at most one job of each class not in J' – call these remaining jobs R_i. For scheduling the jobs in R_i we assume that we have an additional $\varepsilon/3$ units of space in each slot – this is third phase where we have extra speed. We order the jobs in ascending order of their class. We schedule these jobs in the earliest possible slots. Note that a job of processing time $(1+\varepsilon/3)^k$ will finish within $\frac{3}{\varepsilon} \cdot \sum_{j=0}^{k}(1+\varepsilon/3)^j$ time of its release date. Hence the flow time of this job is at most $\left(\frac{1+\varepsilon}{\varepsilon^2}\right)$ times its processing time.
As in the previous section we can bound the total processing time of jobs in our assignment by $4 \cdot \Delta'(x^*) + 2T$ (recall $\alpha = 1$). Again we assume that $T \geq \mathsf{opt}$. Since $\Delta'(x^*) \leq \mathsf{opt}$, we get that the total processing time in our schedule is at most $6\mathsf{opt}$. This implies that the total flow time of the jobs not in J' is at most $O\left(\frac{1+\varepsilon}{\varepsilon^2}\right) \cdot \mathsf{opt}$ which together with the above claim implies the following theorem.

Theorem 2. *The total flow time of jobs in the final schedule is at most* $O(\varepsilon^{-2}\mathsf{opt})$.

We now show how the upper bound can be improved to $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon} \cdot \mathsf{opt})$. It suffices to give a tighter bound for the flow-time of the jobs in R_i for all i. Indeed, the flow-time of the jobs in J' is only opt/ε (Claim 3). Let $P(R_i)$ be the total processing time of the jobs in R_i (in our schedule). Clearly $\sum_i P(R_i) \leq P \leq \mathsf{opt}$. Let P_i be the total processing time of the jobs in J' scheduled on machine i. Define ℓ as the smallest non-negative integer such that $P_i \leq \frac{(1+\varepsilon/3)^\ell}{\varepsilon^2}$. Let x be a constant such that $(1+\varepsilon/3)^x = 1/\varepsilon^2$. Note that x is $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$.
We divide the jobs in R_i into three parts, depending on their class on machine i – *small*, *medium* and *large*. A job in R_i is small if it of class less than ℓ, medium if its class is between ℓ and $\ell + x$, and large if it is of class larger than $\ell + x$. We now bound the flow-time of jobs in each of these parts.

Claim. The total flow-time of small jobs in R_i is at most $O(P_i/\varepsilon)$.

Proof. Assume $\ell > 0$, otherwise we have nothing to prove. Consider a small job of class k in R_i. As argued in the proof of Theorem 2, we can show that the flow-time this job is bounded by $O\left(\frac{(1+\varepsilon/3)^{k+1}}{\varepsilon^2}\right)$. So the total flow time of small jobs in R_i can be bounded by $O\left(\frac{1}{\varepsilon^2} \cdot \sum_{j=0} \ell(1+\varepsilon)^\ell\right)$, which is at most a constant times $\frac{(1+\varepsilon/3)^\ell}{\varepsilon^3}$. But $P_i > \frac{(1+\varepsilon/3)^{\ell-1}}{\varepsilon^2}$. This proves the result. ■

Now we consider medium and large jobs. We use the following fact – the flow-time of a job of class k in R_i can be bounded by

$$P_i + \sum_{j=0}^{k}(1+\varepsilon/3)^j \leq P_i + O\left(\frac{(1+\varepsilon/3)^{k+1}}{\varepsilon}\right) \tag{6}$$

This is so because processing of such a job can be interrupted by either the jobs in J' or those in R_i of smaller class.

Claim. Total flow-time of medium jobs is $O(x \cdot P_i + P(R_i)/\varepsilon)$.

Proof. We use equation (6). Consider a job j of class k. Its processing time on machine i, $p_{ij} = (1+\varepsilon/3)^k$. Therefore, its flow-time can be bounded by $P_i + O(p_{ij}/\varepsilon)$. Summing over all medium jobs, we see that their total flow-time is $x \cdot P_i$ plus $O(1/\varepsilon)$ times the total processing time of medium jobs. ■

Claim. The total flow-time of large jobs is $O(P(R_i)/\varepsilon)$.

Proof. The processing time of a large job is at least $(1+\varepsilon/3)^{\ell+x} \geq \frac{(1+\varepsilon/3)^\ell}{\varepsilon^2} \geq P_i$. So in equation (6), the second term dominates the first term, and so the result follows. ■

Combining the above claims, we get

Theorem 3. *The total flow time of jobs in the final schedule is at most $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon} \cdot \mathsf{opt})$.*

4 Lower Bound for the Parallel Scheduling Problem

In this section, we give lower bound for the approximation ratio of the Parallel Scheduling problem. Recall that in the Parallel Scheduling problem, all machines are indentical. Each job j has the same processing time p_j on all the machines. We shall use the 3-partition problem to prove the hardness of the Parallel Scheduling problem. Recall the 3-partition problem. We are given a set X of $3m$ elements and a bound B. Each element $x \in M$ has size s_x which is an *integer*. We would like to partition the set X into m sets such that the total size of elements in any such set is exactly B. Such a partition is said to be *valid* partition. We can assume without loss of generality that the sum of the sizes of the elements in X is exactly $m \cdot B$. It is NP-complete to decide if such a valid partition exists or not [2].
In fact, we can say more about this problem. If we look at the NP-completeness proof of the 3-partition problem (see for example [2]) we can add two more constraints on the problem instance : (i) For every element x, s_x lies between $B/4$ and $B/2$. Hence if there is a valid partition, then every set in this partition must contain exactly three elements of X, and (ii) The bound B is a polynomial in m, i.e., there is a universal constant c such that B is atmost m^c (for all instances on $3m$ elements). The problem remains NP-complete even in this setting. So when we talk about the 3-partition problem in this section, we shall assume that we are talking about this special case.
The following theorem was proved in [].

Theorem 4. *Given an instance $\mathcal{I}$ of the 3-partition problem, parameters T and K, there is a poly-time computable function f, which gives an instance $f(\mathcal{I})$ of the Parallel Scheduling problem. Here T is a multiple of $K \cdot B$ (B is the bound appearing in $\mathcal{I}$). This reduction has the following properties :*

– *There are m machines in $f(\mathcal{I})$, and the job sizes lie in the range $[KB/4, KB/2]$. The total volume of jobs in $f(\mathcal{I})$ is $T \cdot m$.*
– *If $\mathcal{I}$ has a valid partition, then all jobs in $f(\mathcal{I})$ can be finished during $[0, T]$. Further the flow-time of these jobs is $O(m \cdot T)$.*
– *If $\mathcal{I}$ does not have a valid partition, then any scheduling of jobs in $f(\mathcal{I})$ must leave at least K units of unfinished volume of jobs at time T.*

Now we describe the reduction from the 3-partition problem to the Parallel Scheduling problem. Fix an instance $\mathcal{I}$ of the 3-partition problem. We construct an instance $g(\mathcal{I})$ of the Parallel Scheduling problem. This instance will have ℓ phases, each of length T. Each phase shall invoke Theorem 4. The parameter T in the theorem will remain unchanged, but K will vary. So we shall denote by $f(\mathcal{I}, K)$ the instance returned by the theorem above when the inputs to f are $\mathcal{I}$, T and K. The parameter ℓ will be m^r, for a large enough constant r.
Let us describe phase i. In phase i we get the instance $\mathcal{I}'_i = f(\mathcal{I}, B^{\ell-i})$. All jobs in this instance releasing at time t get released at time $(t+i\cdot T)$ in $g(\mathcal{I})$, i.e., we just shift the release dates by $i \cdot T$ units of time. Let T_ℓ denote the time at which phase $\ell-1$ (the last phase) gets over, i.e., $\ell \cdot T$. Starting from time T_ℓ, we release m jobs of length 1 at each time step t, where $t = T_\ell, \ldots, T_M$, $T_M >> T_\ell$. This completes the construction of the instance $g(\mathcal{I})$of Parallel Scheduling. We shall call the jobs released in phase i as jobs of *class* $\ell - i$.
Suppose $\mathcal{I}$ has a valid partitioning. Then it is easy to see that jobs in phase i of $g(\mathcal{I})$ can be processed during $[iT, (i+1)T]$. Similarly, after time T_ℓ, each new job gets done in 1 unit of time. So the total flowtime is $O(m \cdot T \cdot \ell) + m \cdot (T_M - T_\ell)$, which is close to $m \cdot T_M$.
Now suppose $\mathcal{I}$ does not have a valid partitioning. We would like to argue that for any scheduling of the jobs in $g(\mathcal{I})$, the total flowtime is $\Omega(T_M \cdot \ell)$. This will give a gap of ℓ/m in the approximation ratio.
Consider a fractional schedule S for the jobs in $g(\mathcal{I})$ – a fractional schedule is essentially a fractional packing which obeys the release date. We say that the fractional schedule is *valid* if it satisfies the following condition for every i, $0 \leq i \leq \ell - 1$ – at least $B^{\ell-i}$ volume of jobs of class $\ell - i$ is processed after phase i ends, i.e., after time $(i+1) \cdot T$. Theorem 4 says that every integral schedule is valid. We say that a valid schedule is *minimal* if it satisfies the following stronger property : for every i, $0 \leq i \leq \ell - 1$, there is a job of class $\ell - i$ which has $B^{\ell-i}$ amount of processing left at time $(i+1) \cdot T$.
Let $W(S)$ denote the jobs which are waiting at time T_ℓ in this schedule. Let $F_\ell(S)$ denote the flow-time of S if we start counting from time T_ℓ. In other words, if $N_t(S)$ is the number of jobs waiting at time t in S, then $F_\ell(S) = \sum_{t \geq T_\ell} N_t$. Since the flow-time after T_ℓ is going to dominate the total flow-time for any schedule, we should look for schedules which minimize this component of the flow-time.
The following lemma clarifies the need for a minimal schedule.

Lemma 3. *Given any valid schedule S, there exists a (valid) minimal schedule S' such that $F_\ell(S') \leq 2 \cdot F_\ell(S)$.*

Proof. Fix an i. Let $T_i = (i+1) \cdot T$, the time at which phase i ends. Let $W_i(S)$ denote the jobs of class $\ell - i$ waiting at time T_i. Let j_i be the job in $W_i(S)$ with the latest release date. Let j' be some other job in $W_i(S)$. Suppose we process δ amount of j_i before T_i in some time slot. We now process δ amount of j' in this slot, and leave an extra δ amount of job j_i waiting at time T_i – this can always be done as long as δ is small enough. This extra volume of job j_i is processed after time T_i in the slots in which j' was being processed.
We repeat this process as long as possible. When we stop, either all of j_i is waiting at time T_i, or $W(S)$ has only one job, namely j_i. In either of these cases, we satisfy the conditions of a minimal schedule for this class. We do this for all values of i. Let S' be this schedule.
Clearly S' is minimal. Let $N_i(t)$ be the number of jobs of class $\ell - i$ waiting at time t in S, and let $N'_i(t)$ be the corresponding quantity in S'. Consider $t \geq T_i$. Then $N'_i(t) \leq N_i(t) + 1$ – indeed we are only increasing the processing of j_i beyond T_i. Further if $N'_i(t) = N_i(t) + 1$, then it must be the case that $N_i(t) \geq 1$ (because j_i is replacing the slot at time t occupied by some other job of class $\ell - i$). So we conclude that after time T_i, $N'_i(t) \leq 2N_i(t)$. Summing this over all values of i and for $t \geq T_\ell$, we get the desired result. ■

Theorem 5. *Given any minimal schedule S for the instance $g(\mathcal{I})$, the flow-time of S is $\Omega(T_M \cdot \ell)$.*

Proof. We shall build a schedule S' from S such that $F_\ell(S') \leq F_\ell(S)$ but $W(S')$ will have jobs from each class. This will yield the desired result. However S' may not be a valid schedule and may not even obey released dates.
We will construct S' in ℓ iterations. At the end of iteration i, the following invariants will hold for the schedule S' :

(i) $F_\ell(S') \leq F_\ell(S)$.
(ii) There is at least one job from each of the classes $1, 2, \ldots, i$ with remaining processing time at least 1 in $W(S')$.
(iii) Jobs from classes $i+1, \cdots, \ell$ obey release dates.
(iv) Jobs from class $1, 2, \ldots i$ are processed after time $(\ell - i) \cdot T$.
(v) For each $q \in \{i+1, \ldots, \ell\}$, there is at least one job of class q which has B^q volume of processing left at time $(\ell - q + 1) \cdot T$.

We initialize S' to S – recall that jobs of class i are released during $[(\ell - i) \cdot T, (\ell - i + 1) \cdot T]$. Clearly the invariant holds at $i = 0$ because S is a minimal schedule. Suppose the invariants hold after iteration $i, i \geq 0$. We need to show that they hold at the end of iteration $i+1$ as well. Let us consider S' at the end of iteration i. Suppose $W(S')$ has a job of class $i+1$. Then we do not do make any changes in iteration $i+1$. Clearly the invariants hold at the end of iteration $i+1$ as well. So assume $W(S')$ does not have any job of class $i+1$ at the end of iteration i. The rest of the discussion looks at S' at the end of iteration i.
We know that at time $(\ell - i) \cdot T$ at least B^{i+1} volume of jobs of class $i+1$ are unfinished (using invariant (v) and $q = i+1$). This volume of job must get processed during $[(\ell - i) \cdot T, \ell \cdot T]$ because $W(S')$ does not contain any job from this class. But the total volume of jobs released

during $[(\ell - i)\cdot T, \ell\cdot T]$ equals $m\cdot i\cdot T$ and all these jobs are processed after time $(\ell - i)\cdot T$ (invariant (iv)). So at least B^{i+1} volume of jobs of class i or lower must be waiting at time T_ℓ.

We claim there is a class $q \in \{1,\ldots,i\}$ such that $W(S')$ has at least $\frac{B}{2}\cdot B^q + 1$ volume of class q jobs. Indeed suppose this were not the case. Then the total remaining processing time of jobs from classes $1,\ldots,i$ in $W(S')$ is at most $B/2\cdot \left(B^i + \cdots + B^1\right) + q$ which is at most B^{i+1}. But this contradicts the fact that at least B^{i+1} volume of jobs of class i or lower must be waiting at time T_ℓ.

So suppose such a class q exists. Let J_q be the set of jobs of class q in $W(S')$. There is a job $j \in J_q$ which has remaining processing time at least 1 (using invariant (ii)). Let us consider the jobs in $J_q - \{j\}$. These jobs have remaining processing time at least 1 because j can have size at most $\frac{B}{2}\cdot B^q$. We form a new set of jobs J' as follows. J' is initially empty. We start adding jobs from $J_q - \{j\}$ to J' – we stop as soon as the remaining processing times of jobs in J' exceeds 1. Note that the total remaining processing times of jobs in J' is at most $1 + \frac{B}{2}\cdot B^q \le B^{i+1}$. Invariant (v) implies that there is a job j of class $i+1$ which is waiting at time $(\ell - i)\cdot T$ in S'. Further this job has B^{i+1} amount of processing remaining at time $(\ell - i)\cdot T$. We process all the jobs in J' in the slots in which j is getting processed during $[(\ell - i)\cdot T, \ell\cdot T]$ – note that we will be able to finish all of J' because the processing required is at most B^{i+1} and at least B^{i+1} amount of j gets processed during this period (also observe that we may be violating the release date condition for these jobs – but our invariant conditions are not getting violated). Now we process the extra volume of job j after time T_ℓ in the slots in which jobs in J' were getting processed. Note that this cannot increase $F_\ell(S')$ (because we are replacing all the slots occupied by J' after time T_ℓ by a single job). This ends iteration $i+1$. It is easy to see that all the invariants hold at the end of this iteration.

But now $F_\ell(S') \ge \ell\cdot(T_M - T_\ell)$ because the ℓ jobs remaining at time T_ℓ will keep on waiting till time T_M (each of these jobs has remaining processing time more than 1). The theorem follows since $T_M >> T_\ell$. ∎

We have seen that if the instance $\mathcal{I}$ of 3-partition has a valid partitioning, then $g(\mathcal{I})$ has flow-time $O(m\cdot T_M)$. If $\mathcal{I}$ does not have a valid partitioning, then $g(\mathcal{I})$ has flow-time $\Omega(\ell\cdot T_M)$ (using Lemma 3 and Theorem 5. So it is NP-hard to get an $O(\ell/m)$-approximation algorithm for the Parallel Scheduling problem. Let us express this ratio in terms of the largest processing time P (the lowest processing time is 1). $P \le B/2\cdot B^\ell \le B^{\ell+1}$. We know that $B = m^c$ for some constant c and $\ell = m^r$ for a large enough constant r. So we get $\ell/m = m^{r-1}$. We need to express m^{r-1} in terms of P. Now $P \le m^{cm^r + c} \le m^{rm^r}$ which implies that m^r is $\Omega(\log P/\log\log P)$. So m^{r-1} is $\Omega\left(\frac{\log P}{\log\log P}\right)^{1-\varepsilon}$, where $\varepsilon = 1/r$ is an arbitrarily small constant. But this is same as $\Omega(\log P)^{1-\varepsilon'}$ for an arbitrary small constant ε'.

References

1. Sanjeev Khanna Amit Kumar Chandra Chekuri, Ashish Goel. Multiprocessor scheduling to minimize flow time with epsilon resource augmentation. In *ACM Symposium on Theory of Computing*, pages 363–372, 2004.
2. M.R. Garey and D.S. Johson. *Computers and intractability - a guide to NP-completeness*. W.H. Freeman and Company, 1979.
3. Naveen Garg and Amit Kumar. Better algorithms for minimizing average flow-time on related machines. In *ACM Symposium on Theory of Computing*, pages 730–738, 2006.
4. Naveen Garg and Amit Kumar. Minimizing average flow-time : Upper and lower bounds. To appear in IEEE Foundations of Computer Science, 2007.
5. Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. In *ACM Symposium on Theory of Computing*, pages 110–119, 1997.
6. R. Sitters. Minimizing flow-time for unrelated machines. In *Personal Commnunication*, 2008.