Minimizing Average Flow-time : Upper and Lower Bounds

Naveen Garg
naveen@cse.iitd.ac.in

Amit Kumar
amitk@cse.iitd.ac.in

Indian Institute of Technology Delhi, India

Abstract

We consider the problem of minimizing average flow time on multiple machines when each job can be assigned only to a specified subset of the machines. This is a special case of scheduling on unrelated machines and we show that no online algorithm can have a bounded competitive ratio. We provide an $O(\log P)$-approximation algorithm by modifying the single-source unsplittable flow algorithm of Dinitz et.al. Here P is the ratio of the maximum to the minimum processing times.

We establish an $\Omega(\log P)$-integrality gap for our LP-relaxation and use this to show an $\Omega(\log P/\log\log P)$ lower bound on the approximability of the problem. We then extend the hardness results to the problem of minimizing flow time on parallel machines and establish the first non-trivial lower bounds on the approximability; we show that the problem cannot be approximated to within $\Omega(\sqrt{\log P/\log\log P})$.

1. Introduction

We consider the problem of scheduling jobs that arrive over time in multiprocessor environments. This is a fundamental scheduling problem and has many applications, e.g., servicing requests in web servers. The goal of a scheduling algorithm is to process jobs on the machines so that some measure of performance is optimized. A natural measure is the average flow time of the jobs. Flow-time of a job is defined as the difference between its completion time and its release time, i.e., the time it spends in the system.

The special case of this problem when all machines are identical has received considerable attention in the recent past [1, 2, 12]. There has been recent progress on the problem where machines can have different speeds [7, 6]. However very little is known about the problem when a job can go only on a subset of machines. Often in an heterogeneous environment, e.g., grid computing, a job can only be processed by a subset of machines. This can happen due to various reasons – geographical proximity, capabilities of machines, security reasons. In this paper, we consider scheduling in such heterogeneous environments.

We give new results on approximating the average flow-time of jobs on multiple machines. We consider the following setting : we have several machines of the same speed. Further each job may specify a subset of machines on which it can be processed. We use the following nomenclature : if each job can be processed on any machine, we call it the Parallel Scheduling problem. If we add the constraints that each job can only be processed on a subset of machines, then we call it the Parallel Scheduling with Subset Constraints problem.

We consider both the on-line and the off-line settings. We allow jobs to be preempted. Indeed, the problem turns out to be intractable if we do not allow preemption. Kellerer et. al. [10] showed that the problem of minimizing average flow time without preemption has no online algorithm with $o(n)$ competitive ratio even on a single machine. They also showed that it is hard to get a polynomial time $O(n^{1/2-\varepsilon})$-approximation algorithm for this problem. So preemption is a standard, and much needed assumption when minimizing flow time. It is desirable that our algorithms give non-migratory schedules, i.e., a job gets processed on only one machine. Our scheduling algorithm has this property and the approximation ratio holds even if we compare with migratory schedules. Our lower bound results apply to migratory schedules as well.

Table 1 summarizes our results and the previous results known for the problem of minimizing average flow-time. Here P refers to the ratio of the highest processing requirement to the smallest processing requirement of a job.

Our first result a strong lower bound on the competitive ratio of any on-line algorithm for the Parallel Scheduling with Subset Constraints problem. One might think that this is because of the fact that the set of subsets of machines specified of the jobs is a complex set system. However we show that our lower bounds hold even if this set system is laminar.

In the offline setting, we give a polynomial time logarithmic approximation algorithm for the Parallel Scheduling with Subset Constraints problem. We also show an

	Online		Offline	
	Lower bounds	Upper bounds	Lower bounds	Upper bounds
Parallel Scheduling	$\Omega(\log P)$ [12]	$O(\log P)$ [12]	$\Omega\left(\sqrt{\frac{\log P}{\log\log P}}\right)$	$O(\log P)$ [12]
Parallel Scheduling with Subset Constraints	unbounded		$\Omega\left(\frac{\log P}{\log\log P}\right)$	$O(\log P)$

Table 1. Lower and upper bounds on the approximation ratio for the problem of minimizing average flow-time. The uncited results appear in this paper.

almost matching lower bound on the approximation ratio of any polynomial time algorithm for this problem. Our techniques then extend to give a non-trivial lower bound on the approximation ratio for the Parallel Scheduling problem. This solves an important open problem in the area of scheduling algorithms.

Related Work: The problem of minimizing average flow time (with preemption) on identical parallel machines has received much attention in the past few years [12, 2, 3, 1]. Leonardi and Raz [12] showed that the Shortest Remaining Processing Time (SRPT) algorithm has a competitive ratio of $O(\log(\min(\frac{n}{m}, P)))$, where n is the number of jobs, and m is the number of machines. A matching lower bound on the competitive ratio of any on-line (randomized) algorithm for this problem was also shown by the same authors. It is also interesting to note that this is also the best known result in the off-line setting of this problem. Garg and Kumar [7, 6] recently gave on-line algorithms for minimizing average flow-time on related machines with poly-logarithmic competitive ratio.

Much less is known when we impose the constraint that a job can be scheduled only on a subset of machines. Note that this is a special case of scheduling on unrelated machines [11, 8]. However, nothing is known for minimizing average flow-time in either of these settings.

Proving hardness of approximation for minimizing average flow-time on parallel machines was a long standing open problem of considerable interest. Our paper gives the first non-trivial result in this direction.

Our techniques: Our algorithm for the Parallel Scheduling with Subset Constraints problem establishes an interesting connection of this problem to routing unsplittable flow from a source to multiple sinks in a graph so as to minimize edge congestion. We solve a linear programming relaxation of our problem and then round the fractional solution into a feasible non-migratory schedule using a modified version of an algorithm of Dinitz et.al.[4] for the unsplittable flow problem. The modifications that we have to make to the unsplittable flow algorithm are not straightforward and are necessitated by the presence of empty spaces in the solution to the LP-relaxation.

Our hardness results are based on the idea of obtaining strong integrality gaps for the natural linear programs for these problems. These integrality gap instances then guide us towards constructing hard instances of these problems.

2. Preliminaries

We consider the problem of minimizing average flow-time of jobs on multiple machines. There are m machines, numbered from 1 to m, of the same speed. Each job j has a processing requirement p_j, $1 \leq p_j \leq P$, and a release date, r_j. In the problem Parallel Scheduling with Subset Constraints, each job specifies a subset $S(j)$ of machines on which it can be processed.

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. The jobs can either arrive in an on-line manner or their release dates can be specified off-line. A useful way of thinking about the total flow-time of a schedule is as follows : let $W(t)$ denote the set of *active jobs* at time t, i.e., jobs which have been released before t but not finished at time t. The the total flow time is the integral over time t of $W(t)$.

We give an integer programming formulation for the Parallel Scheduling with Subset Constraints problem [6]. For each job j, machine i, 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.

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

$$\sum_j x_{i,j,t} \leq 1 \text{ for machines } i \text{ and time } t \quad (2)$$

$$\sum_i \sum_t x_{i,j,t} = p_j \quad \text{for all jobs } j \quad (3)$$

$$x_{i,j,t} = 0 \quad \text{if } t < r_j \text{ or } i \notin S(j) \quad (4)$$

$$x_{i,j,t} \in \{0,1\} \text{ for all jobs } j, \text{ machines } i \text{ and time } t \quad (5)$$

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 [6].

Fact 2.1 *The total flow-time of $\mathcal{S}$ is at least $\sum_j \Delta_j(x^{\mathcal{S}})$.*

3. Lower bounds for on-line scheduling

In this section, we show strong lower bounds on the competitive ratio of on-line algorithms for the Parallel Scheduling with Subset Constraints problem. For simplicity, we shall show these lower bounds for deterministic algorithms only, but they can be easily generalized to randomized algorithms with oblivious adversaries.

We first show that there is a strong lower bound even for the Parallel Scheduling with Subset Constraints problem even when the processing requirement of each job is 1.

Theorem 3.1 *There does not exist an on-line algorithm with a bounded competitive ratio for the Parallel Scheduling with Subset Constraints problem even when all job sizes are 1.*

Proof. Let $\mathcal{S}$ be a deterministic on-line algorithm for this problem. Our instance has three machines, which we call 1, 2 and 3. There are two types of unit size jobs. Type I jobs can go on machines 1 and 2 only, while type II jobs can go on machines 2 and 3 only. At each time step $t = 0, 1, \ldots, T-1$, 2 jobs of type I and 2 jobs of type II are released. Note that at time T, at least T jobs must be left in $\mathcal{S}$. Two cases arise – (i) at least $T/2$ jobs of type I remain at time T in $\mathcal{S}$, or (ii) at least $T/2$ jobs of type II remain at time T in $\mathcal{S}$.

Suppose case (i) happens. We release 2 jobs of type I at each time step $t = T, \ldots, L$, where $L >> T$. Observe that in $\mathcal{S}$, we will have $T/2$ jobs waiting at each time step during $[T, L]$. So the total flow-time is $\Omega(T \cdot L)$. However we claim that the total flow-time of the off-line optimum is only $O(L)$. The off-line algorithm schedules type I jobs during $[0, T]$. So at time T, we have T jobs of type I remaining. During $[T, 3T]$, these jobs get processed on machine 3. Note that after time $3T$, there will be no waiting jobs. Assuming $L >> T^2$, we see that the total flow-time is $O(L)$. The argument when case (ii) happens is similar. ■

Recall that $S(j)$ denotes the subset of machines on which a job can get processed. Let $\mathcal{M}$ denote the collection of these sets, i.e., the set system $\{S(j)\}$. It might happen that if the sets in $\mathcal{M}$ have some simple structure, we can get good on-line algorithms. One of these properties can be *laminar structure*. We say that the sets in $\mathcal{M}$ are laminar if any two sets in $\mathcal{M}$ are either disjoint or one of them is a subset of the other. Note that the set systems constructed in the proof of Theorem 3.1 are not laminar. Our next theorem, however, shows that there are strong lower bounds on the competitive ratio even when the set system $\mathcal{M}$ is laminar.

Theorem 3.2 *Consider the Parallel Scheduling with Subset Constraints problem when the set system $\mathcal{M}$ contains only two sets, one contained in another. Then any on-line algorithm must have competitive ratio of $\Omega(P)$.*

Proof. Fix a deterministic on-line algorithm $\mathcal{S}$. There are two machines, which we call 1 and 2. There are two types of jobs – type I jobs have unit size and can go on any machine. Type II job has size $P >> 1$, and can go on machine 2 only. On each time step $t = 0, \ldots, T-1$, we release two jobs of type I. Further, at each time step $t = 0, P, 2P, \ldots, T-1$, we release one job of type II (assuming $T-1$ is a multiple of P). Here $T >> P$. Note that the total volume released by time T is $3T$. Since there are only 2 machines, one of the following cases must happen in $\mathcal{S}$ – (a) at least $T/2$ jobs of type I remain at time T, or (b) at least $T/2P$ jobs of type II remain at time T.

Suppose case (a) occurs. Then we release 2 jobs of type I at each time step $t = T, \ldots, L$, $L >> T$. Note that the total flow-time of $\mathcal{S}$ is $\Omega(T \cdot L)$. But the optimum off-line algorithm has total flow time of $O(T \cdot L/P)$. The off-line algorithm finishes all jobs of type I during $[0, T]$. So it has only T/P jobs (of type II) waiting at each time step during $[T, L]$.

Now suppose case (b) occurs. We release one job of type II at each time step $t = T, T+P, T+2P, \ldots, L$, $L >> T$. Clearly, there is a always a queue of size $T/2P$ jobs in $\mathcal{S}$ during $[T, L]$. So $\mathcal{S}$ pays $\Omega(T \cdot L/P)$. However we claim that the off-line optimum pays only $O(L)$. The off-line algorithm finishes all jobs of type II during $[0, T]$. It has $2T$ jobs of type I waiting at time T. It finishes these jobs during $[T, 3T]$ on machine 1. So after time $3T$, it does not have any queue of waiting jobs. Hence its flow-time is $O(L)$ only. We get the desired result by making T at least P^2. ■

4. Algorithm for Parallel Scheduling with Subset Constraints

We say that a job is of *class* k if $2^{k-1} \leq p_j < 2^k$. Let J_k denote the jobs of class k and let C be the number of classes. 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 $\Delta'_j(x) = \sum_i \sum_t x_{i,j,t} \left(\frac{t-r_j}{\lceil p_j \rceil} + \frac{1}{2} \right)$. 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. It is easy to see that this modification does not increase the cost of a solution x and reduces it by at most a factor 2 so that Fact 2.1 still holds. 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).

By the slot (i, t) we mean the time slot $(t, t+1)$ on machine i.

4.1. The Structure of the Algorithm

We begin with a description of the overall structure of the algorithm. The algorithm proceeds in C iterations. In the k^{th} iteration the algorithm starts with a feasible solution to the LP, x^{k-1} (x^0 is the same as x^*).

Consider a slot (i, t) and define the *available space* in this slot as the total volume of this slot that is either empty or is occupied by jobs of class k in the solution x^{k-1}. In this iteration the algorithm rearranges the jobs of class k in the total available space, so that each job $j \in J_k$ is completely scheduled on one machine in $S(j)$. We denote this new solution to the LP by x^k and argue that $\sum_{j \in J_k} \Delta'_j(x^k) \leq \sum_{j \in J_k} \Delta'_j(x^{k-1}) + U$ where $U = \sum_j p_j$ is the total processing time of all jobs. Since in iteration k we only move jobs of class k, the above implies that $\Delta'(x^k) \leq \Delta'(x^{k-1}) + U$. Thus after C iterations we would have ensured that every job j is scheduled completely on one machine in $S(j)$. Further, $\Delta'(x^C) \leq \Delta'(x^*) + C \cdot U$.

The solution x^C defines a preemptive, non-migratory schedule, $\mathcal{S}$. If some slot in $\mathcal{S}$ is partly empty and a job j can be brought forward to this slot then we do that. Let $\mathcal{S}^f$ be the schedule so obtained and x^f be the LP solution corresponding to this schedule. Note that $\Delta'(x^f) \leq \Delta'(x^C)$. Further, as in [6], we can bound the flow time of $\mathcal{S}^f$ by $\Delta'(x^f) + C \cdot U$. Putting everything together we get that the flow time of schedule $\mathcal{S}^f$ is at most $\Delta'(x^*) + 2C \cdot U$. Since both U and $\Delta'(x^*)$ are lower bounds on the optimum flow time we obtain a schedule whose flow time is atmost $1 + 2 \log P$ times the optimum.

4.2. The k^{th} iteration

We construct a graph G_k for iteration k.

1. G_k has one vertex $v(i, t)$, for each choice of machine i and time t. Besides these, there is one vertex, $v(j)$ for each job j of class k and a vertex $u(t)$ for each time t. The source is a vertex s.

2. There is an edge from $v(i, r_j)$ to $v(j)$ iff $i \in S_j$. This edge has flow $\sum_t x^{k-1}_{i,j,t}$. If this flow is zero we remove the edge. There are no other edges incident at vertex $v(j)$. Note that for each job j the LP has a constraint $\sum_i \sum_t x_{i,j,t} = p_j$. This implies that the total flow reaching vertex $v(j)$ is exactly p_j and this is the demand of vertex $v(j)$.

3. For every i, t there is an edge from $v(i, t)$ to $u(t)$. The flow on this edge is the extent to which the time slot t on machine i is empty, ie $1 - \sum_j x^{k-1}_{i,j,t}$. We refer to such edges as "null-edges" and to the vertex $u(t)$ as a null-vertex. The flow reaching vertex $u(t)$ equals the total empty space in the slots at time t. This is also the demand of vertex $u(t)$.

4. There is an edge from $v(i, t-1)$ to $v(i, t)$ and the flow on these edges is given by conservation constraints. The vertex $v(i, 0)$ is the same as the vertex s.

From the construction of the graph it follows that each edge has a unique predecessor edge.

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 flow from s to the sink $u(t)$ could use multiple paths. Besides this, we have the following additional requirement on the flow f^*:

1. Let e be a null-edge and e' its predecessor. If $f^*(e) > 0$ then $f^*(e') \leq f'(e')$.

2. For any edge, e, $f^*(e)$ should not exceed $f'(e)$ by more than $2^k - 1$.

The unsplittable flow f^* in graph G_k, 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. Thus if the edge $(v(i, r_j), v(j))$ has non-zero flow in f^* then we schedule job j, which is of class k, on machine i. Further, a flow of p units on the null-edge from $v(i, t)$ to $u(t)$ is viewed as scheduling a null-job, having release time t and processing time p (same as p empty slots) on machine i. These jobs are scheduled in the available space for this iteration. To do the actual assignment of jobs to slots on machine i, we order slots by decreasing time and the jobs by decreasing release time; If a job j and a null-job have the same release time then the null-job appears before j in the ordering. The

jobs and slots are considered in this order and each job is assigned to slots with total available space equal to the processing time of the job. Note that we might not be able to assign all jobs to slots since we might run out of available space. It might also happen that only a part of a job can be assigned.

Let x' be a (possibly infeasible) solution to the LP that captures the above reassignment of class k jobs. Note that x' is identical to x^{k-1} for jobs of all classes other than k.

Lemma 4.1 $\sum_{j \in J_k} \Delta'_j(x') \leq \sum_{j \in J_k} \Delta'_j(x^{k-1})$.

Proof. Since we have only assigned class k jobs to the available space for this iteration, the difference in cost can be viewed as arising from the placement of empty spaces. We now argue that in x', the empty spaces only occur later than in the x^{k-1}. Suppose the total empty space at time t in the solution x^{k-1} is p. In the graph G_k, vertex $u(t)$ would have a demand p. This demand p is perhaps routed on multiple paths and leads to the creation of multiple null-jobs each with a release time t. Consider the null-job so created on machine i. The flow f^* we compute has the property that $f^*(e) \leq f'(e)$ where e is the edge from $v(i, t-1)$ to $v(i, t)$. This implies that the total processing time of all jobs (null and of class k) with release time t or more that are scheduled on machine i is at most the total available space at and after time t on machine i in this iteration. Hence the null-job in consideration would be assigned slots on machine i after time t. Since this is true for each null-job created for the empty space at time t, we have established that the empty spaces in x^{k-1} only move ahead in time in the solution x' This proves the lemma. ■

However, the solution x' is not feasible. There are two sources of infeasibility stemming from the fact that in the unsplittable flow we exceed the original flow value on the edges.

Note that on some machine i we might not be able to assign all jobs given by the unsplittable flow in the available space. This is because the flow on the edge $(s, v(i, 1))$ exceeds the initial flow. Note that the sum $\sum_{j \in J_k} \Delta'_j(x')$ does not include the contribution of jobs which do not get assigned. However, the total volume of jobs which are to be scheduled on machine i but cannot be assigned to the available space is at most $2^k - 1$.

The second source of infeasibility could be that a job is scheduled before its release date.

Claim 4.2 *On any machine i, at most $2^k - 1$ units of available space before time t are assigned to jobs released at or after t.*

Proof. The flow through the edge $(v(i, t-1), v(i, t))$ in f^* exceeds the original flow through this edge by at most $2^k - 1$. Since the total available space after t equals the original flow through this edge, the amount of available space needed, before t, to schedule jobs released at or after t, is at most $2^k - 1$. ■

The above two infeasibilities can therefore be easily handled if we create $2^k - 1$ new slots on each machine at the end of the schedule and shift the class k jobs and the empty spaces in the solution x' to the right by $2^k - 1$ units of available space. Let x^k be a feasible solution to the LP which corresponds to this modification.

Claim 4.3 $\sum_{j \in J_k} \Delta'_j(x^k) \leq \sum_{j \in J_k} \Delta'_j(x') + U$.

Proof. We begin by assuming that all empty spaces in x' are also occupied by jobs of class k. For a machine i consider a time t. As a result of the above shift, the total volume of class k jobs scheduled on i after time t would increase by exactly $2^k - 1$. This implies that the contribution of the class k jobs increases by the processing time of all jobs (and any empty spaces) on machine i. Now note that every unit of empty space, which we treated as occupied by class k jobs, would move at least $2^k - 1$ units to the right. Hence, removing the contribution of these empty spaces would lead to a net increase which is at most the total processing time on machine i. Since this is true for all machines the claim follows. ■

This completes our description of the k^{th} iteration and also establishes that the solution to the LP, x^k, obtained at the end of this iteration satisfies, $\Delta'(x^k) \leq \Delta'(x^{k-1}) + U$. We now discuss how to compute the an unsplittable flow meeting the two requirements.

4.3. Computing the Unsplittable Flow

We will modify the unsplittable flow algorithm of Dinitz et.al.(DGG algorithm) to get a flow which satisfies the above requirements.

We begin with a brief review of the DGG algorithm. The algorithm starts with a fractional flow from the source to the sinks. Each sink gets flow equal to its demand which could be routed along multiple paths. The algorithm makes the flow unsplittable by redistributing it along "alternating cycles". An alternating cycle is composed of alternate "forward" and "backward" paths; flow is increased along the backward paths and decreased along the forward paths. A backward path is formed by starting at a sink and following incoming edges till we reach a vertex which has more than one outgoing edge. We take the outgoing edge not on the backward path to start a forward path. The forward path continues by taking outgoing edges till we reach a sink at which point we start building a backward path by taking an incoming edge other than the one on the forward path. At

any point in the algorithm, if a vertex has a terminal and an incoming edge with flow exceeding the demand of this terminal , then we move the terminal across this edge and reduce the flow on the edge by the demand. Once an edge has zero flow it is removed from the graph. This implies that every sink has atleast two incoming edges. We can similarly ensure that the source has at least two outgoing edges and these two facts together imply that we can always build an alternating cycle.

The extent of augmentation along an alternating cycle is determined by the minimum of two quantities — the flow on an edge on the forward paths and the increase in flow on a backward edge that would cause a terminal to move. An edge (u, v) is *singular* if v has at most one outgoing edge. Note that all edges on backward paths are singular and hence flow is increased only on singular edges. Since we do not add any edges to the graph, a singular edge remains singular till it is deleted. Key to the analysis of the algorithm is the fact that we move at most one terminal along a singular edge. This implies that the total unsplittable flow routed along an edge exceeds the initial flow on the edge by the maximum demand.

Our first modification to the DGG algorithm is:

> We call a null-edge carrying zero flow an *empty edge*. An empty edge, say from $v(i,t)$ to $u(t)$ is removed iff the edge from $v(i,t)$ to $v(i,t+1)$ is on a backward path.

Note that the flow on an edge is increased only when it is on the backward path of an alternating cycle. The above modification preserves the property that all such edges are singular. Hence, as in the DGG algorithm we can argue that for any edge, e, $f^*(e)$ exceeds $f'(e)$ by at most the maximum demand sent through e after e is on a backward path. Our second modification will ensure that the maximum demand we send through an edge after it is part of a backward path is 2^{k-1}. This would then imply that our second requirement on f^* is met.

Since the flow corresponding to the null vertex $u(t)$ does not have to be routed unsplittably, we do not have to wait till one of the edges incident to $u(t)$ has flow equal to the demand of $u(t)$. We can route a part of this demand earlier. This gives us our second modification to the DGG algorithm:

> The terminals $u(t)$ are never moved. However, if the flow on a null-edge e, incident to $u(t)$, equals the flow on the preceding edge e', we route this amount of flow from the source to $u(t)$ along the unique path terminating in edge e.

Note that after we route the flow as above, edges e, e' are removed from the graph. The above modification allows us to prove the following useful claim.

Claim 4.4 *Let e be a null-edge carrying non-zero flow. None of the edges on the unique path from s to e was ever on a backward path.*

Proof. Let e' be the immediate predecessor of e. If e' was ever on a backward path then e would also be on this path. At that point the flow on e and e' was the same and hence we would have routed part of the demand of terminal $u(t)$ from the source thereby removing edges e and e'. This implies that e' was never on a backward path and hence none of the edges preceding it could have been on a backward path. ■

This claim lets us prove that flow f^* meets the two requirements.

1. If we route non-zero flow along a null-edge e in the flow f^* then by the above claim, the edge preceding e, say e' was not part of a backward path till the point in time that the flow from s to the null-vertex was routed. At this point the edge e' was removed. Hence, we never increased flow through edge e' and so the final flow through e' does not exceed the initial flow and the first requirement on the flow f^* is met.

2. If we route non-zero flow along a null-edge e in the flow f^* then none of edges preceding e on the path from s to e had been part of a backward path before this point. This implies that once an edge is part of a backward path, it can only carry flow for the terminals $v(j)$. Since the maximum demand for these terminals is $2^k - 1$ and since only one such terminal can travel across this edge (as in the DGG algorithm) we route at most $2^k - 1$ units of flow through the edge after it is part of a backward path. This implies that the final flow through any edge exceeds the initial flow by atmost $2^k - 1$ and establishes our second requirement on f^*.

Since we might not move terminal $u(t)$ even when one of the edges incident to it carries flow equal to its demand, we might now have a situation where a null-vertex has only one edge entering it. Hence if we reach this vertex on a forward path, we will not be able to leave it and so will not be able to build an alternating cycle.

The structure of the problem allows us to get around this last remaining difficulty.

Lemma 4.5 *The modified algorithm is able to find an alternating cycle at each step.*

Proof. Consider all null-vertices which have only one incoming edge and let $u(t)$ be the one with maximum t. Let $(v(i,t), u(t))$ be the only edge entering $u(t)$. Since this edge has non-zero flow none of the edges preceding it would have been part of a backward path. This implies that every null-vertex $u(t'), t' \leq t$ has an edge from $v(i,t')$.

We start our alternating path from vertex $u(t)$, take the edge $(v(i,t), u(t))$ to form a backward path and then continue with a forward path. As we continue this process, suppose, we get stuck at a null-vertex $u(t'), t' < t$. We must have reached $u(t')$ from vertex $v(i,t')$ as we were building a forward path. An alternate way of continuing the forward path would have been to take the path from $v(i,t')$ to $v(i,t)$ and this would have formed an alternating cycle. ■

5. Integrality Gap for Parallel machines

In this section, we show that the integrality gap of the linear program in Section 2 is large. In fact this large gap holds even when we restrict to the special case of the Parallel Scheduling problem. This integrality gap instance will form the basis of our lower bound reductions. Our instance will have m machines and the ratio of the largest job size to the smallest job size will be m as well. The number of jobs n can be much larger than m (in fact has no dependence on m). We pick two parameters C, ℓ such that $C^\ell = m$. We shall make both of these $\Omega\left(\frac{\log m}{\log\log m}\right)$. We shall say that a job is of type k if its size is C^k.

Jobs arrive in ℓ phases, each phase being of length exactly T (T is an arbitrary parameter, it can as big as we wish). Let us describe phase 0 : at each integral time step t, $0 \le t < T$, one job of type ℓ arrives. Phase $i \ge 0$ can be described similarly : at each integral time step t, $i \cdot T \le t < (i+1) \cdot T$, C^i jobs of type $\ell - i$ arrive.

Let us first see what the fractional solution does. In phase i, consider the jobs that arrive at time t. The total volume of these jobs is $C^i \cdot C^{\ell-i} = C^\ell = m$. So the m machines finish these jobs by time $t+1$ (note that in the fractional solution the jobs can be done concurrently on several machines). So every job spends just 1 unit of time in the system. Now note the total processing time of the jobs in our instance is $m \cdot T \cdot \ell$. Therefore, the value of the fractional solution is also $O(m \cdot T \cdot \ell)$.

We now show that any integral solution will have to pay about $\log m / \log\log m$ times this value. The intuition is as follows : consider phase 0. The integral solution cannot finish all jobs in this phase by time T. Indeed, only one machine can be busy at time 0, only two machines can be busy at time 2 and so. So we will need to shift about $m \cdot C^\ell/2$ volume of jobs of type ℓ after time T. The key thing to observe is that this number is a factor of C larger than the amount of *shift* needed in phase 1 and so on. So this shifted volume cannot fit in the gaps created by shifting jobs in subsequent phases. This volume will have to go to the very end of the schedule. We now prove the result formally.

Lemma 5.1 *Consider an integral schedule $\mathcal{S}$. For any i, $0 \le i < \ell$, $\mathcal{S}$ must process at least $mC^{\ell-i}/2$ volume of jobs of type $\ell - i$ after time $(i+1) \cdot T$, which is the time at which phase i ends.*

Proof. Consider phase i. Call a machine at time t *good* if it is processing a job of type $\ell - i$ at time t. We claim that at time $i \cdot T + x$, there are at most $x \cdot C^i$ good machines. Indeed, at each unit of time between $i \cdot T$ and $i \cdot T + x$, C^i jobs of type $\ell - i$ are released. So there are at most $x \cdot C^i$ jobs of type $\ell - i$ at time $i \cdot T + x$ in this schedule. Since a job can be processed by at most one machine at any point of time, the number of good machines at this time must also be at at most $x \cdot C^i$. Let I denote the interval $[i \cdot T, i \cdot T + C^{\ell-i}]$. In this interval, at most $\sum_{x=0}^{C^{\ell-i}} x \cdot C^i$ volume of processing of jobs of type $\ell - i$ is done. This quantity is at most $m \cdot C^{\ell-i}/2$ (because $m = C^\ell$). But the total volume of jobs that can be processed during I is $m \cdot C^{\ell-i}$. So there is at least $m \cdot C^{\ell-i}/2$ volume of time in this interval I during which $\mathcal{S}$ does not process a job of type $\ell - i$. Since the total volume of jobs of type $\ell - i$ released during phase i is exactly $m \cdot T$, at least $m \cdot C^{\ell-i}/2$ volume of such jobs must be processed after this phase ends. ■

We are now ready to prove the main theorem.

Theorem 5.2 *Any integral schedule $\mathcal{S}$ for this instance must have total flow-time at least $\Omega(\min(C, \ell) \cdot m \cdot \ell \cdot T)$.*

Proof. Consider a schedule $\mathcal{S}$. Fix an $i \le \ell/4$. Let T_i denote the time at which phase i ends, i.e., $T_i = (i+1) \cdot T$. Lemma 5.1 says that at least $mC^{\ell-i}/2$ volume of jobs of type $\ell - i$ must be processed after time T_i. Let T' denote the time at which phase $\ell/2$ begins, i.e., $\ell \cdot T/2$. Now one of these two cases must happen in $\mathcal{S}$: either (i) at least $mC^{\ell-i}/4$ volume of jobs of type $\ell - i$ is processed between time T_i and T', or (ii) at least $mC^{\ell-i}/4$ volume of jobs of type $\ell - i$ is processed after time T'.

Suppose case (i) happens. Fix a time $t \in [T', \ell \cdot T]$ ($\ell \cdot T$ is the time at which the last phase ends). During $[T_i, t]$, exactly $(t - T_i) \cdot m$ volume of jobs of type $\ell - i - 1$ or less are released. Since at least $mC^{\ell-i}/4$ volume of jobs of type $\ell - i$ are processed between time T_i and T', it follows that at least this much volume of jobs of type $\ell - i - 1$ or less must be *waiting* at time t in the schedule $\mathcal{S}$. Since each such jobs has processing time at most $C^{\ell-i-1}$, it follows that at least $mC/4$ jobs must be waiting at time t. Since this holds for any time t lying in the interval $[T', \ell \cdot T]$, the total flow time of $\mathcal{S}$ is at least $mC/4 \cdot (\ell \cdot T - T') = \Omega(m \cdot C \cdot \ell \cdot T)$, and so we are done.

Therefore we can assume that case (ii) holds for every $i \le \ell/4$. Fix such an i. Since $mC^{\ell-i}/4$ volume of jobs of type $\ell - i$ translates to at least $m/4$ jobs, at least $m/4$ jobs of type $\ell - i$ finish processing after time T'. So the total flow time of such jobs is at least $m/4 \cdot (T' - i \cdot T)$, which is $\Omega(m \cdot \ell \cdot T)$ because $i \le \ell/4$. But this holds for all values of

i between 0 and $\ell/4$. So the total flow time is $\Omega(m \cdot \ell^2 \cdot T)$. This proves the theorem. ■

Thus we have shown that the integrality gap of this LP in case of parallel machines is $\Omega\left(\frac{\log m}{\log \log m}\right)$.

6. Hardness Results

We now prove lower bounds on the approximation ratio of poly-time algorithms for the Parallel Scheduling and Parallel Scheduling with Subset Constraints problems. Let us first describe the main idea used in these proofs. Recall the integrality gap instance in the previous section. The key reason it worked was as follows : identical jobs are released in a time slot of length T so that the fractional solution could process these jobs in this time slot and *fill* the entire time slot on each machine. However any integral solution would need to process a large volume of such jobs after the slot ends. We achieved this by allowing the fractional solution to process the same job concurrently on multiple machines.

We use a similar idea for constructing lower bound proofs. We show that it is possible to release almost identical jobs in a time slot of length T so that no polynomial time algorithm can distinguish between these two cases : (i) all jobs can be completed in the time slot and hence fill this entire time slot on all machines, or (ii) a significant volume of jobs need to be processed after time T. We use the hardness of the 3-Dimensional Matching problem and the 3-partition problem to prove such results.

6.1. Parallel machine scheduling with subset constraints

Our hardness result will use the 3-Dimensional Matching problem. Recall the definition of the 3-Dimensional Matching problem : we are given three sets X, Y and Z each of size n. We are also given a subset E of $X \times Y \times Z$. We shall call an element of E a *triplet* or a *hyperedge*. The goal is to find a subset E' of E of size n such that each element in $X \cup Y \cup Z$ appears in (exactly) one of these triplets. Such a set of edges is called a perfect matching. The problem of deciding whether an instance of the 3-Dimensional Matching problem has perfect matching is NP-complete [5].

In fact, we shall deal with a special case of the 3-Dimensional Matching problem, which we call the Bounded 3-Dimensional Matching problem. In this problem, the total number of triplets in the set E is at most δ times the size of the set X, where δ is a constant. It turns out that the Bounded 3-Dimensional Matching problem is NP-complete as well [9]. In fact one can prove the following stronger theorem.

Theorem 6.1 *[9] There exists a constant $\gamma > 0$ such that it is NP-hard to distinguish between the following instances of the Bounded 3-Dimensional Matching problem : (i) the instance has a perfect matching, or (ii) any matching in the instance has at most $\gamma \cdot n$ edges.*

Theorem 6.2 *There exist constants $\alpha, \beta > 0$ such that the following statement is true. Let K and T be arbitrary parameters, where T is a multiple of $3K$ (typically $T >> K$). There is a poly-time reduction f from the Bounded 3-Dimensional Matching problem to the Parallel Scheduling with Subset Constraints problem which has the following properties :*

- *Given an instance $\mathcal{I}$ of the Bounded 3-Dimensional Matching problem, $f(\mathcal{I})$ has job sizes in the range $[K, 3K]$. Let m denote the number of machines in $f(\mathcal{I})$. Then the total volume of jobs in $f(\mathcal{I})$ is exactly $m \cdot T$.*
- *Suppose there is a perfect matching in $\mathcal{I}$. Then all jobs in $f(\mathcal{I})$ can be scheduled in a manner so that they finish processing by time T. Further the total flow-time of these jobs is at most $\alpha \cdot m \cdot T$.*
- *Suppose there is no matching in $\mathcal{I}$ which has more than $\gamma \cdot n$ hyperedges. Then any scheduling of jobs in $f(\mathcal{I})$ must leave at least $\beta \cdot m \cdot K$ volume of jobs unfinished at time T.*

Proof. We use the reduction of Lenstra et. al. [11]. Let $\mathcal{I}$ be an instance of the Bounded 3-Dimensional Matching problem. Let E denote the set of triplets, which is a subset of $X \times Y \times Z$. Recall that the sets X, Y, Z have size n each and E has at most $\delta \cdot n$ hyperedges. Let m denote the size of E. We now construct an instance $\mathcal{I}'$ of the Parallel Scheduling with Subset Constraints problem. $\mathcal{I}'$ will not be the instance $f(\mathcal{I})$, but will form the basic building block in the construction of $f(\mathcal{I})$.

In $\mathcal{I}'$ all jobs are released at time 0. There are two kinds of jobs : for each element $v \in X \cup Y \cup Z$, we have one job v of size K (although we are using the same notation for an element of $X \cup Y \cup Z$ and the corresponding job, this should not cause any confusion). There are $m - n$ *dummy* jobs of size $3K$. It is easy to see that the total volume of jobs is $3mK$. There are m machines, one machine for each triplet $e \in E$. A job v can be processed by a machine e if and only if v is in the triplet e. Dummy jobs can be processed on any machine.

Let us see the key properties of $\mathcal{I}'$. Suppose $\mathcal{I}$ has a perfect matching $E' \subset E$. Then on the machines corresponding to E', we can schedule the jobs in these triplets. The remaining $m - n$ dummy jobs go in each of the machines corresponding to $E - E'$. Thus, all jobs in $\mathcal{I}'$ finish by $3K$.

Also the total flow-time of the jobs is at most $3K \cdot (m+2n)$ (because there are $m + 2n$ jobs) which is at most $9mK$.

Now suppose $\mathcal{I}$ does not have a matching of size more than γn. We want to claim that in any scheduling of the jobs in $\mathcal{I}'$, lot of jobs will remain unfinished at time $3K$. Let us see why. Consider a scheduling $\mathcal{S}$ of jobs in $\mathcal{I}'$. We call a machine e corresponding to the triplet (e_x, e_y, e_z) *good* if $\mathcal{S}$ schedules e_x, e_y, e_z on e. Clearly there are at most $\gamma \cdot n$ good machines. Let J_g be the set of jobs from $X \cup Y \cup Z$ which are scheduled on good machines. Recall that there are a total of $3n$ jobs corresponding to $X \cup Y \cup Z$ and at most $3\gamma n$ of these are scheduled on good machines. Let J_b denote the remaining jobs from $X \cup Y \cup Z$. So $|J_b| \geq 3n(1-\gamma)$. Call a machine *bad* if a job from J_b gets scheduled on it. So the total number of bad machines is at least $|J_b|/2$ (because at most 2 jobs from J_b can go on such a machine). Consider a bad machine e corresponding to the triplet (e_x, e_y, e_z). During $[0, 3K]$ either 1 or 2 jobs from J_b get scheduled on it. So one of the following cases must happen : (i) At least K units of time is idle during $[0, 3K]$, or (ii) A dummy job is scheduled on this machine. Note that in case (ii) the machine processes at least K units of volume after time $3K$ (because each dummy job is of size $3K$ and we have at least one job from J_b on this machine). Let M_1 be the bad machines which satisfy (i), and M_2 be the mad machines which satisfy (ii).

If $|M_1| > |M_2|$, then we see that the schedule $\mathcal{S}$ has at least $|M_1|K \geq |J_b| \cdot K/4$ volume of idle time during $[0, 3K]$. Since the total volume of jobs in the instance$\mathcal{I}'$ fits exactly on m machines during $[0, 3K]$ it follows that $\mathcal{S}$ processes at least $|J_b| \cdot K/4$ volume of jobs after time $3K$.

If $|M_1| \leq |M_2|$, then $\mathcal{S}$ processes at least $K \cdot |M_2| \geq |J_b| \cdot K/4$ units of volume after time $3K$. Thus in either case, $\mathcal{S}$ processes $\Omega(K \cdot n)$ volume of job after time $3K$. Since $m = O(n)$, we have shown the theorem when $T = 3K$.

In order to extend it to larger values of T, we simply take $\frac{T}{3K}$ copies of $\mathcal{I}'$ and schedule them one after another. More formally, we construct $f(\mathcal{I})$ as follows. We take $\frac{T}{3K}$ copies of $\mathcal{I}'$. The jobs in the i^{th} copy get released at time $(i-1) \cdot 3K$. This completes the description of $f(\mathcal{I})$. Let us now prove that each of the three statements in the theorem are satisfied.

The first statement is true simply by the properties of $\mathcal{I}'$. Let us prove the second statement. Suppose $\mathcal{I}$ has a perfect matching. Then the jobs in the i^{th} copy can be processed during $[(i-1) \cdot 3K, i \cdot 3K]$. So all jobs finish by time T. Since the flow-time of jobs in any copy is at most $9mK$, the total flow-time is $O(m \cdot T)$. Let us now prove the third statement. Suppose any matching in $\mathcal{I}$ has at most γn hyperedges. Consider the *last* copy of $\mathcal{I}'$ in $f(\mathcal{I})$, i.e., the copy which starts at time $T - 3K$. By properties of $\mathcal{I}'$ above, $\Omega(mK)$ volume of jobs in this copy must be processed after time T by any schedule. Thus the theorem is proved. ■

We are now ready to give the reduction from Bounded 3-Dimensional Matching to Parallel Scheduling with Subset Constraints. The reduction closely follows the construction of the integrality gap example in Section 5. Let $\mathcal{I}$ be an instance of the Bounded 3-Dimensional Matching problem. We shall invoke Theorem 6.2 several times. The parameter T will remain fixed, but K will vary on different calls to this theorem. So let $f(\mathcal{I}, K)$ denote the instance of the Parallel Scheduling with Subset Constraints problem returned by Theorem 6.2 when given the instance $\mathcal{I}$ of the Bounded 3-Dimensional Matching problem and the parameter K.

We now show how to obtain an instance $g(\mathcal{I})$ of the Parallel Scheduling with Subset Constraints problem which will complete the hardness proof. The construction has ℓ phases, each phase having length T. Let us describe phase i. We get the instance $f(\mathcal{I}, C^{\ell-i})$. 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. Here C is a parameter we shall fix later.

Having described $g(\mathcal{I})$ let us see the properties of this reduction. Suppose $\mathcal{I}$ has a perfect matching. Then jobs in phase i can be scheduled during $[i \cdot T, (i+1) \cdot T]$. So the total flow-time is $O(m \cdot T \cdot \ell)$.

Now suppose $\mathcal{I}$ does not have a matching of size more than $\gamma \cdot n$. Consider phase i. Theorem 6.2 implies that at least $\Omega(mC^{\ell-i})$ volume of jobs of size $C^{\ell-i}$ must be waiting at time $(i+1) \cdot T$ in any scheduling of the jobs in $g(\mathcal{I})$. But this is analogous to Lemma 5.1. Therefore we can use the same arguments as in Theorem 5.2 to show that any schedule must have flow-time at least $\Omega(\min(C, \ell) \cdot m \cdot \ell \cdot T)$.

Thus we have shown it NP-hard to get a poly-time $O(\min(C, \ell))$-approximation algorithm for minimizing the total flow-time for the Parallel Scheduling with Subset Constraints problem. Now as in Section 5, set $C = \ell$ such that $C^\ell = m$. So we get $\Omega(\log m / \log\log m)$ hardness result for this problem.

6.2. Parallel Scheduling

We 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 [5].

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 [5]) 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 at most 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 is the analogue of Theorem 6.2 :

Theorem 6.3 *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.*

Remark : The result in this theorem is much weaker than that in Theorem 6.2. Indeed consider the case when $\mathcal{I}$ does not have a valid partition. The amount of volume remaining at time T in the theorem above is much smaller than the size of even a *single* job.

Proof. Fix an instance $\mathcal{I}$ of the 3-partition problem and the parameters K and T. Let X denote the set of items in $\mathcal{I}$ and s_x denote the size of x. Let $|X| = 3m$. Recall that all sizes are integers. Let B be the bound specified in $\mathcal{I}$. We first construct an instance $\mathcal{I}'$ of Parallel Scheduling. There are $3m$ jobs in $\mathcal{I}'$, one job for each element in X. The job corresponding to x has release date 0 and processing requirement $K \cdot s_x$. There are m machines.

Let us see the properties of $\mathcal{I}'$. Clearly job sizes are in the range $(KB/4, KB/2)$. Suppose $\mathcal{I}$ has a valid partitioning. Let $I(j)$ be the set of 3 items assigned to set j in such a partition. Then schedule the jobs corresponding to $I(j)$ on the machine j. Clearly, on each machine we assign a total of KB volume of job. So all jobs finish by time $K \cdot B$. Since there are only three jobs per machine, it is easy to see that the flow-time is also $O(K \cdot B \cdot m)$.

Suppose $\mathcal{I}$ does not have a valid partitioning. We claim that any scheduling in $\mathcal{I}'$ must leave at least K units of unfinished volume at time $K \cdot B$. Suppose this is not true. Let $\mathcal{S}$ be a schedule which contradicts this claim. Let $I(j)$ be the jobs scheduled on machine j by $\mathcal{S}$. If we scale down the sizes of the jobs in $I(j)$ by a factor of K to their actual sizes in $\mathcal{I}$, we see that the set corresponding to j contains elements which have total size less than $B + 1$. But since all quantities are integers, it must be the case that the sum of the sizes s_x of the jobs $x \in I(j)$ is at most B. Since this holds for all machines, and the sum of the sizes of all jobs is $m \cdot B$, we see that the sets $I(j)$ form a valid partitioning of $\mathcal{I}$, a contradiction. This proves the theorem for the case $T = K \cdot B$. For a larger value of T (while maintaining the constraint that T is a multiple of $K \cdot B$), we just take $\frac{T}{KB}$ copies of $\mathcal{I}'$ and argue as in the proof of Theorem 6.2. ■

The rest of the plan is similar to that in Section 6.1. However, since Theorem 6.3 is much weaker than Theorem 6.2, we get a weaker lower bound. There are ℓ phases. Each phase shall invoke Theorem 6.3. The parameter T in the theorem will remain unchanged, but K will vary. So again, 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^2.

Let us describe phase i. In phase i we get the instance $\mathcal{I}'_i = f(\mathcal{I}, m^{2(\ell-i)c})$. Recall that c is such that the bound B is at most m^c for any instance of the 3-partition problem. As before we shift the release dates of the jobs in $\mathcal{I}'_i$ by $i \cdot T$ units. Let us call this instance of Parallel Scheduling as $g(\mathcal{I})$.

Suppose $\mathcal{I}$ has a valid partitioning. Then it is easy that jobs in phase i of $g(\mathcal{I})$ can be processed during $[iT, (i+1)T]$. So all jobs finish by time $\ell \cdot T$ and the total flow-time is $O(m \cdot T \cdot \ell)$. 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 flow-time is much larger than $O(m \cdot T \cdot \ell)$.

The following fact is clear from the proof of Theorem 6.3.

Fact 6.4 *For each i, $1 \leq i \leq \ell$, at least $m^{2(\ell-i)c}$ volume of jobs from $\mathcal{I}'_i$ are scheduled in $\mathcal{S}$ after time $(i+1) \cdot T$, i.e., the time at which phase i ends.*

The proof of the following theorem proceeds along the same lines as Theorem 5.2.

Theorem 6.5 *If $\mathcal{I}$ does not has a valid partitioning, then any scheduling of the jobs in $g(\mathcal{I})$ must incur $\Omega(m^2 \cdot l \cdot T)$ flow-time.*

Proof. Fix a scheduling $\mathcal{S}$ of $g(\mathcal{I})$. Consider an $i < \ell/4$. Let T_i be the time at which phase i ends, i.e., $(i+1) \cdot T$. Let T' be the time at which phase $\ell/2$ begins, i.e., $\ell \cdot T/2$.

Fact 6.4 implies that at least $m^{2(\ell-i)c}$ volume of jobs from the instance $\mathcal{I}'_i$ are scheduled after T_i. Now two cases arise : (i) at least $m^{2(\ell-i)c}/2$ volume of these jobs is processed after T' in $\mathcal{S}$, or (ii) at least $m^{2(\ell-i)c}/2$ volume of jobs from $\mathcal{I}'_i$ is processed between T_i and T' in $\mathcal{S}$.

Consider case (ii) first. Consider a time t between T' and $\ell \cdot T$ ($\ell \cdot T$ is the time at which the last phase ends). Let $\mathcal{I}''_{i+1}$ denote $\mathcal{I}'_{i+1} \cup \cdots \cup \mathcal{I}'_\ell$.

Claim 6.6 *At least $m^{2(\ell-i)c}/2$ volume of jobs from $\mathcal{I}''_{i+1}$ must be waiting at time t.*

Proof. Suppose t lies in phase j, i.e., between $j \cdot T$ and $(j+1) \cdot T$. Recall the construction of $\mathcal{I}'_j$ in Theorem 6.3. In $\mathcal{I}'_j$, jobs get released at $\frac{T}{KB}$ time instances, namely, $j \cdot T, j \cdot T + KB, j \cdot T + 2KB, \ldots$. Further, the total volume of jobs released during $[j \cdot T + rKB, j \cdot T + (r+1)KB]$ is exactly $m \cdot KB$. Now suppose t lies between $j \cdot T + rKB$ and $j \cdot T + (r+1)KB$. Let t' denote the time $j \cdot T + (r+1)KB$. From $j \cdot T$ to t', exactly $(t' - j \cdot T) \cdot m$ volume of jobs from $\mathcal{I}'_j$ get released. Further between T_i and $j \cdot T$, we release exactly $(j \cdot T - T_i)$ volume of jobs from $\mathcal{I}'_{i+1} \cup \cdots \cup \mathcal{I}'_{j-1}$. Thus we see that exactly $(t' - T_i) \cdot m$ volume of jobs from $\mathcal{I}''_{i+1}$ are released between T_i and t'.

Now, if we schedule $m^{2(\ell-i)c}/2$ volume of jobs from $\mathcal{I}'_i$ between T_i and t' (note that t' is larger than T'), then this much volume of jobs from $\mathcal{I}''_{i+1}$ must be waiting at time t'. Since no jobs get released between t and t', the jobs waiting at t' must also be waiting at t. This proves the claim. ■

Now each job in $\mathcal{I}'_{i+1}$ has processing time at most $B \cdot m^{2(\ell-i-1)c} \leq m^{2(\ell-i-1)c+c} \leq m^{2(\ell-i)c}/m^2$ (assuming c is at least 2). So at least m^2 jobs must be waiting at time t. The time t was chosen arbitrarily between $T \cdot \ell/2$ and $T \cdot \ell$. So the total flow time is at least $\Omega(m^2 \cdot \ell \cdot T)$. This proves the theorem.

So it must be the case that (i) happens. Thus for each value of i between 1 and $\ell/2$, at least one job from $\mathcal{I}'_i$ is waiting at time T'. So the total flow-time is at least $\Omega(\ell^2 \cdot T)$ which proves the theorem because $\ell = m^2$. ■

So we have shown that it is NP-hard to get a poly-time $O(m)$ approximation algorithm for this problem. Let us express this in terms of P. Note that $P = B \cdot m^{2 \cdot \ell \cdot c}$. Now $\ell = m^2$. So we get $P \leq m^{2 \cdot m^2 \cdot c + c}$. This implies that m is $\Omega\left(\sqrt{\frac{\log P}{\log \log P}}\right)$.

7. Open Problems

We leave open the problem of minimizing average flow-time on unrelated machines. We believe there is a poly-logarithmic approximation algorithm for this problem. In fact, one can write a linear program for this problem similar to what we have in this paper. We conjecture that the integrality gap of this natural linear program is poly-logarithmic.

8. Acknowledgements

Part of this work was done when both authors were visitng the Max-Planck-Institut fuer Informatik, Saarbruecken. The research of the first author is partly supported by the "Approximation Algorithms" partner group of MPI-Informatik, Germany.

References

[1] N. Avrahami and Y. Azar. Minimizing total flow time and total completion time with immediate dispatching. In *Proc. 15th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 11–18, 2003.

[2] B. Awerbuch, Y. Azar, S. Leonardi, and O. Regev. Minimizing the flow time without migration. In *ACM Symposium on Theory of Computing*, pages 198–205, 1999.

[3] C. Chekuri, S. Khanna, and A. Zhu. Algorithms for weighted flow time. In *ACM Symposium on Theory of Computing*, pages 84–93, 2001.

[4] Y. Dinitz, N. Garg, and M. X. Goemans. On the single-source unsplittable flow problem. In *IEEE Symposium on Foundations of Computer Science*, pages 290–299, 1998.

[5] M. Garey and D. Johson. *Computers and intractability - a guide to NP-completeness*. W.H. Freeman and Company, 1979.

[6] N. Garg and A. Kumar. Better algorithms for minimizing average flow-time on related machines. In *ICALP*, pages 181–190, 2006.

[7] N. Garg and A. Kumar. Minimizing average flow-time on related machines. In *ACM Symposium on Theory of Computing*, pages 730–738, 2006.

[8] K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. In *ACM Symposium on Theory of Computing*, pages 408–417, 1999.

[9] V. Kann. Maximum bounded 3-dimensional matching is max snp-complete. *Information Processing Letters*, 37:27–35, 1991.

[10] H. Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In *ACM Symposium on Theory of Computing*, pages 418–426, 1996.

[11] J. K. Lenstra, D. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machine. *Mathematical Programming*, 46(3):259–271, 1990.

[12] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *ACM Symposium on Theory of Computing*, pages 110–119, 1997.