Non-Preemptive Min-Sum Scheduling with Resource Augmentation

Nikhil Bansal [*] Ho-Leung Chan [†] Rohit Khandekar [*] Kirk Pruhs [‡] Baruch Schieber [*]
Cliff Stein [§]

Abstract

We give the first $O(1)$-speed $O(1)$-approximation polynomial-time algorithms for several nonpreemptive min-sum scheduling problems where jobs arrive over time and must be processed on one machine. More precisely, we give the first $O(1)$-speed $O(1)$-approximations for the non-preemptive scheduling problems

- $1 \mid r_j \mid \sum w_j F_j$ *(weighted flow time),*
- $1 \mid r_j \mid \sum T_j$ *(total tardiness),*
- *the broadcast version of* $1 \mid r_j \mid \sum w_j F_j$*,*

an $O(1)$-speed, 1-approximation for

- $1 \mid r_j \mid \sum \overline{U}_j$ *(throughput maximization),*

and an $O(1)$-machine, $O(1)$-speed $O(1)$-approximation for

- $1 \mid r_j \mid \sum w_j T_j$ *(weighted tardiness).*

Our main contribution is an integer programming formulation whose relaxation is sufficiently close to the integer optimum, and which can be transformed to a schedule on a faster machine.

1 Introduction

In this paper, we give the first polynomial-time $O(1)$-speed $O(1)$-approximation algorithms for nonpreemptive min-sum scheduling problems where jobs arrive over time and must be processed on one machine. In the problems we consider, each job j has a release time r_j and a size p_j. A feasible schedule assigns each job j to run during a time interval $[x_j, C_j]$ of length p_j such that $r_j \leq x_j$ and at any time at most one job is running.

We consider various min-sum objective functions. The *flow* or *response time* of a job is $F_j = C_j - r_j$, the time that job j is in the system before completing. If a job has a deadline d_j, the *tardiness* T_j of a job j is the amount by which the completion of j exceeds its deadline, i.e., $T_j = \max\{C_j - d_j, 0\}$. For each of these objectives we can take a weighted sum over all jobs, and thus obtain total weighted flow time $\sum w_j F_j$ and total weighted tardiness $\sum w_j T_j$. We also consider weighted flow in a *broadcast scheduling* (or *batching*) environment, where requests from clients for variable sized pages arrive over time, and a request for a page is satisfied by the first broadcast of this page that starts after this request arrives [10].

All the problems that we consider are NP-hard [7]. Further, there are no polynomial-time approximation algorithms for these problems with reasonable approximation ratios. The problem of minimizing total (unweighted) flow cannot be approximated to within a factor of $n^{1/2-\epsilon}$ for any $\epsilon > 0$ on an n-job instance, unless P=NP [11]. This hardness result also applies to broadcast scheduling. The problems in which jobs have deadlines, cannot be approximated in polynomial-time to within any finite factor because the problem of determining whether the objective function is 0 (all jobs complete by their deadlines) is itself NP-hard [7].

For each of these problems, we contend that worst-case approximation ratios do not yield the appropriate insights, as they essentially tell us to give up because no algorithm will be good. However, we need to solve these (and more complicated variants of these) problems, need ways to analyze and compare algorithms, and explanations for why algorithms typically do much better than their worst-case bounds.

With this motivation, Kalyanasundaram and Pruhs [9] introduced the following method of analysis: compare the candidate algorithm, equipped with a faster machine, to OPT, the optimal objective with a unit speed machine. Phillips *et al.* [12] named this methodology *resource augmentation analysis*, and defined a s-speed ρ-approximation

[*]IBM T.J. Watson Research, P.O. Box 218, Yorktown Heights, NY. {nikhil,rohitk,sbar}@us.ibm.com.
[†]Computer Science Department, University of Pittsburgh. hlchan@cs.pitt.edu.
[‡]Computer Science Department. University of Pittsburgh. kirk@cs.pitt.edu. Supported in part by NSF grants CNS-0325353, CCF-0514058 and IIS-0534531.
[§]Department of IEOR, Columbia University, New York, NY. cliff@ieor.columbia.edu. Supported in part by an IBM faculty fellowship.

algorithm to be one which, using a machine of speed-s, can achieve an objective function value no more than $\rho \cdot \text{OPT}$. They also defined the analogous notion of using extra machines instead of/in addition to extra speed: an m-machine s-speed ρ-approximation algorithm is one which, using m speed-s machines can achieve an objective function value no more than $\rho \cdot \text{OPT}$ where OPT is the optimal objective function value on a single unit speed machine.

In the last decade, resource augmentation has gained wide acceptance (for a survey of the many resource augmentation results in scheduling see [14]). Often, resource augmentation analysis tends to identify scheduling algorithms that perform well in practice. Several intuitive explanations for this phenomenon have been proposed. Here we give a "sensitivity" explanation. For concreteness, we use the problem $1 \mid r_j \mid \sum F_j$, although this motivation applies to all problems we consider. The proof that it is NP-hard to approximate $1 \mid r_j \mid \sum F_j$ to within a factor of $\sqrt{n}$, uses a reduction from 3-partition. In the created instances for $1 \mid r_j \mid \sum F_j$, many very small jobs are released every roughly B time units, where B is the size of each set in the 3-partition problem. To create an optimal schedule, the intervening time periods must each be packed with 3 jobs that sum to exactly B. In fact, one actually gets a $\sqrt{n}$ gap in the objective between the optimal schedule and any other feasible schedule. Inspecting the proof, one sees that the job sizes are very carefully tuned. Informally, if the job sizes are perturbed even slightly, the hardness proof falls apart. Perturbing the job sizes is essentially equivalent to perturbing the speed of the machine. If by changing the speed by a small amount, we can approximate the objective function well, this is evidence that the hard instances must be tuned to the particular speed of the machine. So intuitively, the existence of an s-speed $O(1)$-approximation algorithm indicates that hard instances need to be carefully tuned and thus are rare in practice.

For the problem $1 \mid r_j \mid \sum F_j$, Phillips *et al.* [12] give an $O(\log n)$-machine $(1+\epsilon)$-approximation polynomial time algorithm. While this result might reasonably be viewed as positive, it is somewhat unsatisfying for two reasons: the resource augmentation is not constant, and the algorithm is quite naive as it merely uses one machine for all jobs of about the same size.

The obvious open question is then whether there is an $O(1)$-speed $O(1)$-approximation algorithm for $1 \mid r_j \mid \sum F_j$. The main reason why this question remained open for the last decade is that the obvious polynomial-time computable lower bounds are too weak. For non-preemptive scheduling problems, the most natural lower bounds come from preemptive schedules and from linear relaxations of exact integer programming formulations. It is known that preemptive schedules on slower processors can be much better than nonpreemptive schedules on faster processors, thus making the preemptive schedule an unacceptable choice for a lower bound in any analysis. More precisely, Phillips *et al.* [12] show that there exists an input instance I such that the optimal nonpreemptive flow given one speed-s machine, where $s \leq n^{1/4-\epsilon}$ and $\epsilon > 0$, is polynomially larger than the optimal preemptive flow given one unit speed machine.

There are several natural integer linear programming formulations for scheduling problems. (See e.g. [15, 8, 2].) All of the standard formulations have the shortcoming that the linear relaxation with a slower processor can have much better objectives than the optimal integer objective with a faster processor, once again making them useless for a resource augmentation analysis. For intuition, consider the standard strong-time-indexed formulation of $1 \mid r_j \mid \sum F_j$. In this formulation, a variable x_{jt} indicates whether job j starts running at time t. A straightforward exact integer linear program would minimize the objective $\sum_{j \in J} w_j F_j$, where the flow for job j is $F_j = \sum_t (t + p_j - r_j) x_{jt}$. One would then need to add obvious constraints to ensure that only one job is run at each time, and that every job is run. Now consider an instance where big jobs of size L arrive every $2L$ time units, and small jobs of size 1 arrive every other time unit. The optimal integer objective, even with a $\Theta(1)$ speed processor, is $\Theta(LT)$, where T is the duration of the instance. However, by scheduling a big job to extent $1/2$ every L time units and a small job to extent $1/2$ during each time step leads to a solution to the relaxed linear program with objective value $\Theta(T)$.

1.1 Our Results

In this paper we give the first $O(1)$-speed $O(1)$-approximations for the non-preemptive scheduling problems

- $1 \mid r_j \mid \sum w_j F_j$ (weighted flow time),
- $1 \mid r_j \mid \sum T_j$ (total tardiness),
- the broadcast version of $1 \mid r_j \mid \sum w_j F_j$,

An $O(1)$-speed, 1 approximation for

- $1 \mid r_j \mid \sum \overline{U}_j$ (throughput maximization),

and an $O(1)$-machine, $O(1)$-speed $O(1)$-approximation for

- $1 \mid r_j \mid \sum w_j T_j$ (weighted tardiness).

In Section 2, we explain our techniques in some detail for the problem of $1 \mid r_j \mid \sum w_j F_j$, where we finally obtain a 12-speed 2-approximation algorithm. In Section 3, we sketch how to extend these techniques to work for the other problems. In section 4 we show that our approach cannot give a $(2-\epsilon)$-speed, $n^{o(1)}$-approximation for flow time. In contrast, we do show that an $(1+\epsilon)$-speed, O(1)-machine, O(1)-approximation algorithm exits. In these results, we do not try to make the constants as small as possible, rather our aim is to explain a general technique applicable to many problems.

All our algorithms round the relaxation of a novel integer programming formulation that incorporates an additional lower bound on F_j, which is the main technical contribution of this paper. The intuition for this additional lower bound is simple. Consider the case of a job k that the LP decides to schedule in the interval $[t, t+p_k)$. Then, any job j which is released in the interval $[t, t+p_k)$ must start no earlier than time $t+p_k$, and thus must have a flow F_k of at least $p_j + (t + p_k - r_j)$. We then add such constraints to our linear programming formulation. To see the benefit of this, consider the case when j is a large job and k is a small job. Suppose that we need to move k to run after j. In the original time indexed LP, we have nothing to "charge" this move to, as k has not incurred any flow in the linear program. But, with the addition of the new lower bound on F_k, we can now charge the move against this lower bound. Observe that the resulting integer program is no longer exact, in fact even if we solved the integer program exactly, we would only have a 2-approximation.

1.2 Related Results

The only min-sum objective which has $O(1)$-approximation algorithms in the non-preemptive case with release dates is weighted completion time $1 \mid r_j \mid \sum w_j C_j$ [13, 1]. The fact that the total completion time objective $\sum w_j C_j$ doesn't take into account the release dates is the reason that good approximation algorithms are possible, and is also the reason that the objective is of limited practical use.

There has been a long line of work on $O(1)$-approximation algorithms for throughput maximization, and more generally for scheduling with job interval constraints [16, 4, 3, 6]. However, these do not imply a 1-approximation using $O(1)$-machine with $O(1)$ speed, as the "natural" LP cannot give such a result. Chuzhoy *et al.* [5] consider the problem of minimizing the number of machines needed to find a feasible schedule for jobs with release time and deadline constraints. This result can be viewed as a resource augmentation since it gives an $O(m^2)$-machine polynomial time algorithm for scheduling a feasible instance of an m-machine problem. Chuzhoy *et al.* achieve the result by a clever LP formulation that forbids certain intervals for some jobs. Our throughput maximization algorithm uses a similar idea (in fact the constraints in our LP are identical to that of [5] for the single machine case[1]).

At a high level, incorporating the second contribution term in the LP is related to the idea of "forbidden intervals" used by Chuzhoy *et al.* However, the problem settings and metrics are very different. For instance, for flow time minimization there are no deadlines; a job can be scheduled anywhere after its release time and hence forbidden intervals cannot be defined.

2 The Algorithm for $1 \mid r_j \mid \sum w_j F_j$

In this section, we prove our main theorem.

Theorem 1 *There exists a 12-speed $(2+\epsilon)$-approximate deterministic polynomial-time algorithm for $1 \mid r_j \mid \sum F_j$, and a 12-speed 4-approximate deterministic polynomial-time algorithm for $1 \mid r_j \mid \sum w_j F_j$.*

We first give a randomized pseudo-polynomial-time 12-speed 2-approximation algorithm. Then we explain how to derandomize the algorithm, and create a polynomial-time algorithm. In converting to a polynomial-time algorithm we lose the additional approximation factors. We use $\text{OPT}(I)$ to denote the optimal flow time for an instance I. The steps in our algorithm are:

1. The original instance J is modified to create a new instance $\hat{J}$. In $\hat{J}$ the job sizes are rounded down so that the possible job sizes form a geometric sequence.

2. From $\hat{J}$, a linear program LP is created. An integer solution to LP can be interpreted as an aligned schedule. An *aligned* schedule is one in which each job with size p is started at a time that is an integer multiple of p. The optimal solution to LP will be a lower bound to $\text{OPT}(J)$.

3. The linear program LP is then solved. An arbitrary solution is then converted into a canonical solution that essentially favors jobs that are released earlier.

4. The solution of LP is randomly rounded into a pseudo-schedule. In a pseudo-schedule each job is run exactly once, but more than one job may be running at each time.

5. Using some additional speed, this pseudo-schedule is converted into a feasible schedule for $\hat{J}$.

6. Finally, again using some additional speed, a feasible schedule for J is produced.

The correctness should be apparent as we now describe the steps in greater detail.

[1]Chuzhoy *et al.* [5] do not consider the throughput maximization version, but for the single machine case their rounding algorithm can also be seen to imply an $O(1)$-machine, 1-approximation, and an $O(1)$-speed, 1-approximation.

$$\begin{array}{lrcll} \min & \sum_{j\in J} w_j F_j & & & \\ \text{s.t.} & & & & \\ & \sum_t x_{jt} & = & 1 & \forall j \in J \\ \sum_{j\in J} & \sum_{\tau:\tau\in(t-p_j,t]} x_{j\tau} & \leq & 1 & \forall t \in \mathbb{Z} \\ & \forall j \in J: \quad F_j & = & \frac{1}{2}\left(\sum_t (t+p_j-r_j)x_{jt} + p_j + \sum_{k:C_k^{-1}>C_j^{-1}} \sum_{t\in[r_j-p_k+1,r_j]} (t+p_k-r_j)x_{kt}\right) & \\ & x_{jt}, F_j & \geq & 0 & \forall j \in J, t \in \mathbb{Z} \end{array} \qquad (1)$$

2.1 Rounding Job Sizes

Let $\beta > 1$ be an integer to be determined later. In the instance $\hat{J}$, the size of job j is $\hat{p}_j = \frac{1}{2}\beta^{\lfloor \log_\beta p_j \rfloor}$, that is, the new job size is half the largest power of β not greater than p_j.

Lemma 2 *There is an aligned schedule for* $\hat{J}$ *with* $\sum w_j F_j \leq \text{OPT}(J)$.

Proof: Given the optimal weighted flow schedule $\text{OPT}(J)$ on J, we construct an aligned schedule on $\hat{J}$. Consider a job j scheduled to start at time t in $\text{OPT}(J)$. Since $\hat{p}_j \leq p_j/2$, there exists a time $s \in [t, t+p_j/2]$ that is integer multiple of $\hat{p}_j$. Job j starts at time s in our aligned schedule. Since $\hat{p}_j \leq p_j/2$, j completes in this aligned schedule for $\hat{J}$ before j completes in $\text{OPT}(J)$. ∎

To simplify the notation from here on, we use p_j to denote $\hat{p}_j$. We scale the time axis by a factor of 2 and assume that the sizes of jobs p_j are integer powers of β, i.e., $p_j \in \{1, \beta, \beta^2, \ldots, \beta^\kappa\}$ for some integer κ. Let $\mathcal{C}_i = \{j \in J \mid p_j = \beta^i\}$ and call jobs in $\mathcal{C}_i$ as *class-i jobs*. Let C_j^{-1} be the class of job j. We call an interval $[t, t+\ell)$, where t is an integer multiple of ℓ, as an *aligned ℓ-interval*.

2.2 The Linear Programming Formulation

We consider strong-time-indexed formulations where there is a variable x_{jt} for each job j and for each time t, which indicates whether job j starts at time t. Since we are seeking aligned schedules, there will only exist variables x_{jt} for times t that are an integer multiple of p_j. Recall the additional lower bound on the flow time implied by the observation that if a job k is running at time r_j, then job j must wait until job k completes before job j can begin. That is, the flow time is at least p_j plus the time between r_j and when job k finishes, which in our notation is $p_j + \sum_{k\in J\setminus\{j\}} \sum_{t\in[r_j-p_k+1,r_j]} (t+p_k-r_j)x_{kt}$.

For technical reasons, we will apply this lower bound only when job j is in a smaller class than job k. We are now ready to give the linear program (1) that we will use. We recall that even the integer program does not solve the scheduling problem exactly.

The assignment constraints in the first two lines guarantee that every job is scheduled, and that at most one job is run at each time. Let $LP^* = \{x^*_{jt}, F^*_j\}$ be the (fractional) optimum solution to (1).

Lemma 3 *The optimal LP objective* $\sum_{j\in J} w_j F^*_j \leq \text{OPT}(J)$.

Proof: As both $\sum_t (t+p_j-r_j)x_{jt}$ and $p_j + \sum_{k:C_k^{-1}>C_j^{-1}} \sum_{t\in[r_j-p_k+1,r_j]} (t+p_k-r_j)x_{kt}$ lower bound the flow time for job j, the average of these two terms lower bounds the flow time for job j. ∎

We now show that LP^* defines a laminar schedule for each class C_i that favors high weighted jobs. We also can convert LP^* to a canonical schedule that breaks ties in favor of earlier arriving jobs. This property will be crucial in producing the pseudo-schedule. We say a job j *runs* at time t if x^*_{jt} is positive. A job *starts* at the earliest time that it runs, and *completes* at the last time that it runs. The *lifetime* of a job is the interval between when it starts and when it completes. Let $X(t,j) = \sum_{s\leq t} x_{js}$ be the extent to which LP^* has processed job j up until time t. A schedule is *canonical* if for any two jobs i and j, with $p_i = p_j$, $w_i = w_j$ and $r_i < r_j$ (or $r_i = r_j$ and $i < j$), job j does not run in LP^* earlier than the completion time for job i. By making local exchanges, it is very easy to modify LP^* to be canonical in polynomial time, without altering the value of the objective. We introduce a new relation $\preceq_i$ on jobs in class C_i, defined as follows: $j \preceq_i k$ if and only if

$$(1)\ w_j < w_k,\ \text{or}\ (2)\ (w_k = w_j) \wedge (r_k < r_j),\ \text{or}$$
$$(3)\ (w_k = w_j) \wedge (r_k = r_j) \wedge (k \leq j)\ .$$

It is easily seen that $\preceq_i$ is a total order. The next two easy lemmas (proofs omitted) describe structure in the ordering of the jobs. Let $\preceq$ denote the union of the orders $\preceq_i$ for all job classes.

Lemma 4 *In LP^*, for any two jobs $j, k \in \mathcal{C}_i$ that are in the same class, if $j \preceq k$, then there can not exist a time τ such that $0 < X(\tau, k) < 1$ and j runs at time τ.*

This implies that

Lemma 5 (Laminar schedule) *In LP^*, consider two distinct jobs $j, k \in \mathcal{C}_i$ in the same class. Assume there exists a time τ_1 such that $0 < X(\tau_1, j) < 1$ and job k runs at time τ_1. Then the earliest time $\tau_2 \geq \tau_1$ that job j can be run next is the completion time for job k.*

2.3 Creating the Pseudo-Schedule

We construct the pseudo-schedule by applying α-*rounding* to each class separately. In α-rounding we pick an offset $\alpha \in [0, 1)$ uniformly at random, then starting from time 0, we schedule a job in each class whenever the LP has first scheduled a cumulative $\alpha, 1 + \alpha, 2 + \alpha, \ldots$ units of jobs. Formally, to obtain the schedule for class C_i, let $X(t) = \sum_{j \in C_i} \sum_{s \leq t} x_{js}$ be the extent to which LP^* has processed jobs in C_i up until time t. For each non-negative integer h, let t_h be the time when $X(t)$ first exceeds $h + \alpha$. We schedule a class-i job j at time t_h. The job j is chosen as follows: Let $\gamma = h + \alpha - X(t_h - 1)$. Consider all the jobs $j' \in C_i$ such that $x_{j' t_h} > 0$ and order them as $j_1, j_2, \ldots, j_q$ in the order determined by the laminar schedule condition at time t_h (which is descending $\preceq$ order). The job j is the job j_k in this order such that $x_{j_1 t_h} + \ldots + x_{j_k t_h}$ first exceeds γ. In our procedure above, we use the same α for each class (note that α is still uniformly distributed in $[0, 1)$ for each class). The next two lemmas state useful properties of this pseudo-schedule.

Lemma 6 *The pseudo-schedule output by the above rounding procedure satisfies the following properties:*

1. *Each job $j \in J$ is scheduled exactly once (obviously it is scheduled in an aligned p_j-interval).*
2. *No two jobs from the same class $\mathcal{C}_i$ are scheduled in the same aligned β^i-interval.*
3. *Consider any aligned β^i-interval for $0 \leq i \leq \kappa$. The total size of all the jobs in classes $\mathcal{C}_0, \ldots, \mathcal{C}_i$ scheduled in this interval is at most $\beta^i + \frac{\beta^{i+1}-1}{\beta-1} < \beta^i(2 + \frac{1}{\beta-1})$.*

Proof: The first property follows directly from the laminar property of the schedule of jobs within a class and the nature of α-rounding. Assume to reach a contradiction that some job $j \in \mathcal{C}_i$ was scheduled at distinct times t_1 and t_2 with $t_1 < t_2$, and consider the time interval I between when j is run by LP^* at time t_1 and at time t_2. Then by Lemma 4 and Lemma 5, any job $k \in C_i$ run by LP^* during the time interval I must satisfy $j \preceq k$, and k's lifetime must be contained in the interval I. Thus the extent to which LP^* runs jobs in $C_i - \{j\}$ during I is an integer. Since the extent to which LP^* runs job in j during I is fractional, this contradicts the assumption that LP^* has processed jobs to an extent of some integer plus α during both the period when j was running at time t_1 and the period when j was running at time t_2.

The second property follows from the "volume" constraint that states that the sum of the fractions of all jobs running at any time t is upper bounded by 1.

For the third property, consider any aligned β^i-interval I. Let $\delta_{i'}$ be the fractional extent to which jobs in $\mathcal{C}_{i'}$ are scheduled in I for $0 \leq i' \leq i$. Summing the volume constraints for all these jobs, we get that $\sum_{i'=0}^{i} \beta^{i'} \delta_{i'} \leq \beta^i$. Furthermore at most $\lceil \delta_{i'} \rceil < \delta_{i'} + 1$ jobs from $\mathcal{C}_{i'}$ can be scheduled in I in the pseudo-schedule. Thus the total volume of jobs in $\mathcal{C}_0, \ldots, \mathcal{C}_i$ that can be scheduled in I is at most $\sum_{i'=0}^{i} \beta^{i'} \lceil \delta_{i'} \rceil \leq \sum_{i'=0}^{i} \beta^{i'} (\delta_{i'} + 1) \leq \beta^i + (\beta^0 + \beta^1 + \cdots + \beta^i) = \beta^i + \frac{\beta^{i+1}-1}{\beta-1}$. ■

The flow time of jobs in the pseudo-schedule can be easily related to their LP contribution.

Lemma 7 *For job $j \in J$, the expected flow time of j in the pseudo-schedule is $\sum_t (t + p_j - r_j) x^*_{jt}$.*

Proof: Consider a job j and time t. As α varies from 0 to 1, there are exactly x^*_{jt} fraction of choices of α for which job j is scheduled at time t in the pseudo-schedule. Since α is chosen uniformly at random, the expected flow time of j is thus $\sum_t (t + p_j - r_j) x^*_{jt}$. ■

2.4 Converting the Pseudo-Schedule into a Feasible Schedule

We use a factor $(2 + \frac{1}{\beta-1})$ speedup to convert the pseudo-schedule into a feasible schedule that schedules at most one job at any single time. Furthermore, we argue that the weighted flow time of the final schedule for J is at most twice the value of the LP relaxation, and hence, at most $2 \cdot LP^*$. Notice that Lemma 7 only uses the first lower bound on flow time. This conversion is precisely where we use our second lower bound on flow time in the LP formulation.

Consider the pseudo-schedule produced above. Call a job *maximal* if it does not overlap with any other job of larger size. In Figure 1, for example, the job in class-4 is maximal. We associate a natural β-ary tree with any maximal job and the jobs overlapping with it. Recall that the pseudo-schedule schedules any class-i job in an aligned β^i-interval. Thus, the tree-nodes in level i correspond to the aligned β^i-intervals overlapping with the maximal job.

We give a procedure that uses a speed up factor of $(2 + 1/(\beta - 1))$, and given a tree corresponding to a maximal job, it feasibly schedules all the jobs in that tree in the

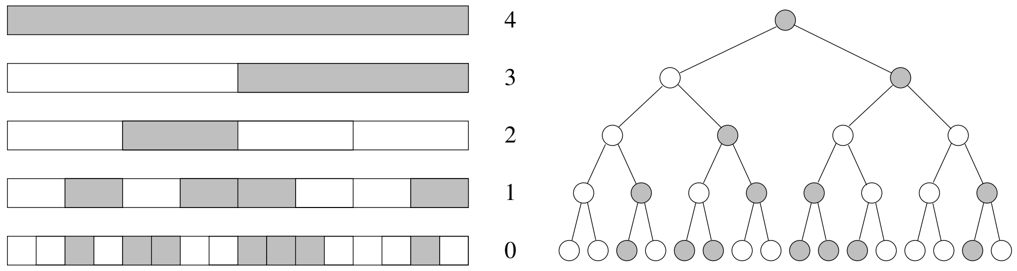

Figure 1. Example of a pseudo-schedule output by the rounding procedure for $\beta = 2$ and its corresponding β-ary tree. The numbers indicate the classes and shaded boxes/tree-nodes correspond to jobs j scheduled in aligned p_j-intervals. The total size of shaded nodes in any sub-tree (containing a node and all its descendants) is at most $(2 + \frac{1}{\beta-1})$ times the size of the root of the sub-tree (see Lemma 6-3).

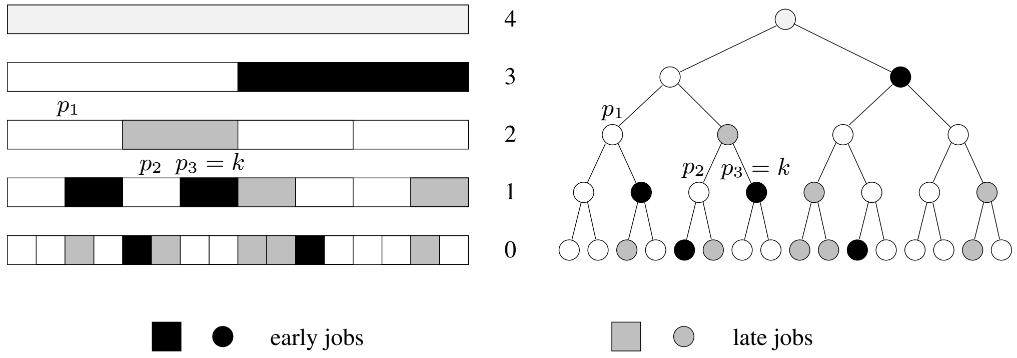

Figure 2. Partition of jobs into early and late jobs.

β^i-interval corresponding to the root. The schedule is feasible in the sense that each job is scheduled after its release time, and moreover, the flow time of each job can be related to its LP contribution. Since all the maximal jobs are non-overlapping, applying the above procedure to each tree produces a schedule for entire instance.

Consider a maximal job j in class-i scheduled in an aligned β^i-interval $I = [\tau, \tau + \beta^i)$. Let J_I denote the set of jobs scheduled in the interval I, and let T_I denote the β-ary tree associated with the pseudo-schedule in I. We partition the jobs $J_I \setminus \{j\}$ into two sets *early* and *late*. The *early* jobs are the jobs $\{k \in J_I \setminus \{j\} \mid r_k < \tau\}$ that are released before time τ. These jobs can be scheduled anywhere in I. Note that even though an early job k is scheduled during the interval I, it does not "pay" the "penalty term" $(\tau + p_j - r_k)x^*_{j\tau}$ in its flow F^*_k. The *late* jobs are the jobs $\{k \in J_I \setminus \{j\} \mid \tau \le r_k < \tau + \beta^i\}$ that are released in I. A late job k can be scheduled no earlier than its release time r_k. Note that a late job k "pays" the "penalty term" $(\tau + p_j - r_k)x^*_{j\tau}$ in its flow F^*_k.

We now describe our procedure FIT, to convert the pseudo-schedule into a feasible schedule:

Algorithm FIT: We first shrink all jobs in J_I by a factor of $(2 + \frac{1}{\beta-1})$. We then compute the POSTORDER[2] traversal of the β-ary tree T_I. We schedule all early jobs in the order they appear in POSTORDER(T_I). The maximal job j is scheduled after the early jobs. We then compute the PREORDER[3] traversal of T_I. We schedule all late jobs in the order they appear in PREORDER(T_I). The late jobs are then "right-justified", or shifted as far right as possible so that the last late job completes at the end-point of interval I.

The following lemma shows that the schedule computed by the FIT procedure is feasible.

[2]Recall that the POSTORDER traversal of a single-node tree v is defined as POSTORDER$(v) := v$ and that of a β-ary tree T with root r and left-to-right sub-trees $T_1, \ldots, T_\beta$ is recursively defined as POSTORDER$(T) :=$ POSTORDER$(T_1), \ldots,$ POSTORDER$(T_\beta), r$.

[3]Recall that the PREORDER traversal of a single-node tree v is defined as PREORDER$(v) := v$ and that of a β-ary tree T with root r and left-to-right sub-trees $T_1, \ldots, T_\beta$ is recursively defined as PREORDER$(T) :=$ $r,$ PREORDER$(T_1), \ldots,$ PREORDER(T_β).

Lemma 8 *The schedule output by the* FIT *procedure satisfies the following properties.*

1. *The jobs in J_I are scheduled in the interval I such that no two jobs overlap.*
2. *Each early job in J_I completes no later than its completion time in the pseudo-schedule.*
3. *Each late job in J_I starts no earlier than its start time in the pseudo-schedule, and it completes within I.*

Proof: The first property follows from the observations that the total size of all the jobs in J_I is at most $\beta^i(2+\frac{1}{\beta-1})$, the length of the interval I is β^i, and we shrink all the jobs by a factor of $(2+\frac{1}{\beta-1})$.

We now prove the second property. Consider an early job $k \in J_I$. Let its completion time in the pseudo-schedule be $\tau+\tau_k$. It is sufficient to argue that the total size of early jobs (including k) that come no later than k in POSTORDER(T_I) is at most $\tau_k(2+\frac{1}{\beta-1})$ before shrinking. To this end, consider the prefix of POSTORDER(T_I) up to node k. (See Figure 2 for an example, here k is the second early job in class 1). Let $T_1,\ldots,T_q$ be the disjoint subtrees of T_I that are traversed in POSTORDER(T_I) up to node k. Note that the root of T_q is k. Now let $p_1,\ldots,p_q$ be the (disjoint) intervals occupied by the roots of $T_1,\ldots,T_q$. (Again see Figure 2 which shows the intervals p_1, p_2 and p_3, and the corresponding subtrees on the right). Note that the total size of $p_1,\ldots,p_q$ is precisely τ_k. Furthermore, from Lemma 6-3, the total size of jobs in J_I that are contained in intervals $p_1,\ldots,p_q$ is at most $(2+\frac{1}{\beta-1})$ times the total size of these intervals. Thus, in particular, the total size of early jobs in these intervals is at most $\tau_k(2+\frac{1}{\beta-1})$ and the property follows.

The proof of the third property follows from the same argument, now applied to PREORDER(T_I). ∎

The next lemma shows an upper bound on the expected flow time of the final schedule.

Lemma 9 *The expected weighted flow time of the final schedule for $\hat{J}$ output by the* FIT *procedure is at most $2\cdot\sum_{j\in\hat{J}} w_j F_j^* \le 2\cdot \text{OPT}(\hat{J})$.*

Proof: It is sufficient to argue that the expected flow time of a job j is at most

$$2\cdot F_j^* = \left(\sum_t (t+p_j-r_j)x_{jt}^*\right) + \left(p_j + \sum_{k:C_k^{-1}>C_j^{-1}} \; \sum_{t\in[r_j-p_k+1,r_j]} (t+p_k-r_j)x_{kt}^*\right).$$

Let F_j denote the random variable whose value is the flow time of j in the final schedule for $\hat{J}$. Let F'_j denote the random variable whose value is the flow time of j in the pseudo-schedule. Let $\mathcal{E}_j$ be the event that the job j is a late job in the pseudo-schedule. We now observe that

$$\begin{aligned}
\mathbf{E}[F_j] &= \mathbf{E}[F'_j] + \mathbf{E}[F_j - F'_j] \\
&= \sum_t (t+p_j-r_j)x_{jt}^* + \mathbf{E}[F_j - F'_j] \\
&= \sum_t (t+p_j-r_j)x_{jt}^* + \mathbf{Pr}[\overline{\mathcal{E}}_j]\cdot\mathbf{E}[F_j - F'_j \mid \overline{\mathcal{E}}_j] \\
&\quad + \mathbf{Pr}[\mathcal{E}_j]\cdot\mathbf{E}[F_j - F'_j \mid \mathcal{E}_j] \\
&\le \sum_t (t+p_j-r_j)x_{jt}^* + \mathbf{Pr}[\mathcal{E}_j]\cdot\mathbf{E}[F_j - F'_j \mid \mathcal{E}_j].
\end{aligned}$$

The first equality follows from Lemma 7, and the inequality follows from the fact that $\mathbf{E}[F_j - F'_j \mid \overline{\mathcal{E}}_j] \le 0$ which in turn follows from Lemma 8 parts 1 and 2. To complete the proof, we now argue that

$$\mathbf{Pr}[\mathcal{E}_j]\cdot\mathbf{E}[F_j - F'_j \mid \mathcal{E}_j] \le \mathbf{Pr}[\mathcal{E}_j]\cdot\mathbf{E}[F_j \mid \mathcal{E}_j] \le p_j + \sum_{k:C_k^{-1}>C_j^{-1}} \; \sum_{t\in[r_j-p_k+1,r_j]} (t+p_k-r_j)x_{kt}^*.$$

If j is a late job, let k be the random variable denoting the maximal job that overlaps with j in the pseudo-schedule and let τ denote the random variable whose value is the start time of k in the pseudo-schedule. Since j is late for k, it must have been released during $[\tau,\tau+p_k]$ and also scheduled at some time t during $[r_j,\tau+p_k]$ by the pseudo-schedule. Thus the probability that j is late for some job k is at most $\sum_{\tau:\tau\in[r_j-p_k+1,r_j]} x_{k\tau}^*$. Moreover, as j is scheduled at some $t\in[r_j,\tau+p_k]$ by the pseudo-schedule, and it must be scheduled no later than $\tau+p_k$by FIT, its flow time is at most $\tau+p_k-r_j$. Thus the expected flow time of j due to k being the maximal job is at most $\sum_{\tau:\tau\in[r_j-p_k+1,r_j]}(\tau+p_k-r_j)x_{j\tau}^*$, which is exactly twice the contribution (corresponding to the second term) of job k to the flow time of j in the LP formulation. ∎

The aligned schedule that we constructed for $\hat{J}$ is a feasible schedule for J with a 2β faster processor. The FIT procedure required a speed-up of $(2+\frac{1}{\beta-1})$. Thus the overall speedup factor is $2\beta(2+\frac{1}{\beta-1})$ that takes its minimum value 12 when $\beta=2$. Combining the results of Sections 2.1 through 2.4, and as the optimum flow time for J is at least as large as the flow time for the optimum aligned schedule for $\hat{J}$, we obtain the following result.

Lemma 10 *There is a randomized pseudo-polynomial-time 12-speed 2-approximation algorithm for $1 \mid r_j \mid \sum w_j F_j$.*

In the remaining subsections, we show how to derandomize the algorithm and then make it polynomial time.

2.5 Derandomization

Consider a particular aligned β^i-interval I. As we vary α from 0 to 1, there will be at most n changes in the class-i jobs scheduled during I. Further it is easy to efficiently compute the α's where the class-i job scheduled at I changes. Let m be the number of variables in our linear program. Thus we can compute at most nm values where the pseudo-schedule produced by our α rounding changes. By exhaustively trying one α from each of the resulting intervals, we can obtain a deterministic algorithm.

2.6 Making the Algorithm Polynomial Time

The linear program LP, as given, is not polynomial-sized. We give two conversions into a polynomial time algorithm, one for the unweighted case, and one for the weighted case.

We first show that, for the unweighted case, we can solve a polynomial-size linear program, while only increasing the total flow by a $1 + o(1)$ factor. Let $p_{\max} = \max_j p_j$. Then each job j clearly runs somewhere in the interval $L_j = [r_j, r_j + np_{\max}]$. Let $L = \cup_j L_j$ be the set of all times a job might possibly run in any optimal schedule. Clearly the length of L is at most $|L| \leq n^2 p_{\max}$. Now define $p_{\text{small}} = p_{\max}/n^3$ and round all jobs processing times up to be integer multiples of p_{small}. Consider what this does to the optimal schedule. Each job's completion time is increased by at most np_{small} and hence the total flow is increased by $n^2 p_{\text{small}} \leq p_{\max}/n \leq F_{\text{OPT}}/n$. Similarly, we can round up each release date to be a multiple of p_{small}, with the same increase in total flow. The resulting instance now has polynomial size, since the range of processing time is polynomial, as is the total set of times in the set L, and the flow has increased by a $1 + o(1)$ factor.

We now sketch how to obtain a polynomial-time algorithm for the problem $1 \mid r_j \mid \sum w_j F_j$, by approximating our pseudo-polynomial sized LP by an LP of polynomial size.

We begin by replacing sequences of variables of the form $x_{js}, x_{j(s+1)}, x_{j(s+2)}, \ldots x_{jt}$ by one variable of the form $x_{js(t+p_j)}$ if nothing "interesting" happens during the time interval $[s,t]$. Intuitively, since nothing interesting happens during $[s,t]$, we will only lose a factor of 2 in the objective by assuming that each variable x_{ju}, for $u \in [s, t-p_j]$, is equal to $x_{jst} \cdot p_j/(t-s)$, that is the start time of job j is spread evenly throughout the aligned p_j-intervals in $[s,t]$.

We now define what the interesting times are inductively from the shortest possible job sizes to the longest possible job sizes. In our LP there will be one variable of the form x_{jst} for each job $j \in C_i$ for each aligned β^i-interval that contains an interesting time, and for each maximal sequence of aligned β^i-intervals that do not contain any interesting time. Assume that we are currently considering class-i jobs (and have already finished handling the jobs of class $< i$). Let $\hat{r}_j$ be the smallest integer multiple of p_j that is at least r_j. An aligned β^i-interval I of the form $[\hat{r}_j + xp_j, \hat{r}_j + (x+1)p_j]$ for some job j contains an interesting time if one of the following holds:

- x is an integer power of 2, or
- I contains the release time of a job, or
- there is a variable of the form x_{kab} for a job $k \in C_l$ for $l < i$ where either a or b is properly contained in I.

Call a variable of the form x_{jst} where $[s,t]$ is an aligned p_j-interval a *regular variable*, otherwise call x_{jst} a *smeared* variable. Similarly define the time intervals $[s,t]$ to be regular or smeared. Finally if there is a smeared variable of the form x_{jst} for a job in class C_i where $[s,t]$ is properly contained in a smeared interval $[s',t']$ for in a class $i' < i$, then break the smeared interval $[s',t']$ into three smeared intervals by cutting at s and t. As a result of this construction, smeared intervals are either identical or disjoint. Further, it is easy to verify that changing where a job j is run within a smeared interval $[s,t]$ for a smeared variable x_{jst} only changes the flow for that job by at most a factor of 2.

We claim that this leaves only a polynomial number of variables of the form x_{jst}. The objective is $\sum w_j F_j$. We now need to add constraints that each job is run. Let $u_1, \ldots, u_m$ be a collection of times such that there is exactly one u_k for each maximal time interval where no interval $[s,t]$ for a variable x_{jst} starts or finishes. We now add a constraint for each u_k that expresses that the sum of the $(p_j/(t-s))x_{jst}$, for $[s,t]$ that contain u_k, is at most 1. Finally we add the constraint

$$F_j \geq \frac{1}{2}\left(\sum_{r_j \leq s} \sum_{s+p_j \leq t} x_{jst}(t-r_j) + p_j + \right.$$
$$\left. \sum_{k: C_k^{-1} > C_j^{-1}} \sum_{s \in [r_j - p_k + 1, r_j]} (s + p_k - r_j) x_{ks(s+p_k)} \right)$$

Note that no smeared interval can contain the time r_j, by the definition of a regular interval.

We need to argue that for every aligned schedule S on $\hat{J}$ that there is a feasible solution for this LP where the value of the objective is at most a constant factor worse than the weighted flow for S. If S schedules job j to start at time u and $x_{ju(u+p_j)}$ is a regular variable, then we set $x_{ju(u+p_j)}$ to 1. Otherwise say u is part of a smeared interval $[s,t]$ corresponding to the variable x_{jst}. Then we increment x_{jst} by 1. Feasibility follows from the fact that smeared intervals are identical or disjoint. The fact that the objective for the LP is not more than a constant factor higher than the weighted flow for S follows from the fact that we cut at times where the flow for some job crossed an integer power of 2.

We then solve this polynomially sized LP. Then by local exchanges we can modify our LP solution so that it gives a laminar schedule for each class (say by always giving preference to heavier jobs, breaking ties in favor of earlier released jobs). We then use an α rounding to get a pseudo-schedule and proceed as before.

This completes the proof of Theorem 1.

3 Extensions to Other Metrics

In this section, we explain how to extend the techniques of the previous section to other objective functions. As many of the details are similar to those for flow time, we only sketch the results, emphasizing the differences between the new metric and flow time.

3.1 Weighted Tardiness

We can extend our results to the problem of minimizing weighted tardiness. Recall that the tardiness of job j is defined by $T_j = \max(C_j - d_j, 0) = \max(F_j - (d_j - r_j), 0)$. Thus, to extend our results, we need to overcome two technical difficulties. First, we must deal with the fact that we are subtracting $(d_j - r_j)$ from the objective, and second we must deal with taking the maximum with zero. We now sketch how to modify the results of Section 2.

The linear programming formulation will still have variables x_{jt} with the same first two assignment constraints. Instead of the constraint on F_j, we will have the following constraints. Let u_{jt} be the tardiness that job j will incur if started at time t and let y_j be the tardiness that j must necessarily incur because of other higher class jobs k whose execution overlaps with the release time r_j of j, i.e., jobs k with start times t such that $t \leq r_j \leq t + p_k$ and such that $p_k > p_j$. So $y_j = \sum_{k \in J: p_k > p_j} \sum_{t \in (r_j - p_k, r_j]} \max(t + p_k + p_j - d_j, 0) x_{kt}$. We then have the objective function $\sum_j w_j (y_j + \sum_t u_{jt} x_{jt})$. This objective is now at most twice the optimal total weighted tardiness.

We now give two different approaches, depending on whether we are considering weighted tardiness, or the unweighted case when all weights are one.

For unweighted tardiness, the schedule still has a structural property, since within a class, the jobs will run in earliest deadline first (EDF) order. The pseudo-schedule satisfies the properties of Lemma 6, and expected total tardiness of job j is at most $\sum_t u_{jt} x^*_{jt}$. We then perform the same conversion to a schedule, the salient point being that each job either completes earlier than its completion time in the pseudoschedule, or the increase in its completion time can be charged against the new lower bound that we added to the linear program. We then prove the analog of Lemma 9, and see that we have 12-speed 2-approximation algorithm.

For weighted tardiness, we do not have the EDF ordering property. Thus, we must divide the jobs of each class into two sets, an *on-time* set and a *late* set. A job is put into the on-time set if more than half of its fractional weight is scheduled before its deadline and into the late set otherwise. We will then use two machines, one for the late jobs and one for the on-time jobs. For the late jobs in a class, you should always run the highest weighted job first, therefore we establish a "laminar" property. The on-time jobs do not contribute to the objective function, thus we can establish the laminar property by swapping jobs. We then continue as in the unweighted case, converting to a scheduling, again observing that each job either completes earlier than its completion time in the pseudoschedule, or the increase in its completion time can be charged against the new lower bound that we added to the linear program. We then prove the analog of Lemma 9 and see that we have a 2-machine 24-speed 4-approximation algorithm.

3.2 Unweighted Throughput Maximization

In the unweighted throughput maximization problem, each job has a release time and a deadline and the goal is to maximize the number of jobs completed by their deadlines. We add a penalty constraint to the LP formulation that if placing a job j at time t completely subsumes the interval $[r_{j'}, d_{j'}]$ of some job j' then $x_{jt} = 0$. It is easily seen that restricted to jobs in a single class, the LP solution forms a laminar schedule. However we cannot apply the α rounding as previously. The problem is that jobs may not necessarily be scheduled to an extent of 1 by the LP. Hence it could be that some job j of class i is scheduled to a non-zero extent at times t and t', but the cumulative amount of class-i jobs (other than j) scheduled between t and t' is non-integral. This may cause the α rounding to schedule the same job many times.

We use the following observation to get around this problem. Suppose j and j' are jobs in the same class such that j' lies in the subtree of j in the laminar tree (i.e., j is executed both before and after j'), then j can be executed wherever j' is. Thus, we can arbitrarily replace j' by j in the LP solution as long as the total amount of j scheduled does not exceed 1. Applying this idea repeatedly, the solution can be reduced to the form such that ignoring the leaves of the laminar tree, the cumulative amount of jobs in each subtree is integral. We then apply α rounding to these jobs. Since the jobs corresponding to leaves are scheduled at exactly one location by the LP, we can use a separate α rounding for these jobs. Jobs of the same class may now be placed at the same location, however this can be handled using another speed up factor of two. Thus we obtain a 24-speed 1-approximation algorithm for maximizing throughput.

3.3 Broadcast Scheduling

We now explain how to extend our technique to broadcast scheduling. The setting for broadcast scheduling is a collection of pages with sizes. Requests from clients for these pages arrive over time. A request for a page is satisfied by the first broadcast of this page that starts after this request arrives. One can express broadcast scheduling as a strong-time-indexed integer linear program as in [10]. There will be a variable x_{jst} that indicates that broadcasts for page j start at time s, and at time t, but at no time in between. We augment this linear program with the same additional lower bound for the flow as we did previously, and then proceed as before. The only issue that arises is in the FIT procedure. The issue is that some jobs may be both early and late, and thus might have to be scheduled at both the start and the end of the interval. By doubling the speed, thus getting a 24-speed algorithm, we can still obtain a feasible schedule.

4 Other Flow Time Results and Lower Bounds

In this section, we show limits on rounding our linear program, and also show that if we allow multiple machines, we can get a stronger result.

A natural open question raised by our results so far is whether $(1+\epsilon)$-speed $O(1)$-approximation polynomial-time algorithms exist for these problems. We show that such a result can not be obtained from our linear programming formulation, and at least 2-speed is needed for rounding the fractional solution. In particular, even for the unweighted case, we show that

Theorem 11 *There are job sequences for which the total flow time of any non-preemptive schedule on a $(2-\epsilon)$-speed machine is polynomially larger (in the number of jobs) than the value of the optimal fractional solution LP^* (on a speed 1 machine).*

Proof: Without loss of generality we assume that ϵ is small enough that $k = 1/\epsilon$ is an even integer. Consider a job sequence where all jobs have weight 1. Let x be the integer $\lceil 2(\ln 2)/\epsilon \rceil$, and let $L = k^{x+1}$. There is one big job of size L released at time 0. There are x batches of medium jobs, each job of size 1. The i-th batch, for $i = 1, \ldots, x$, consists of $L(1-\epsilon(1-\epsilon/2)^i)$ jobs that are released during $[(i-1)L^2, (i-1)L^2 + L]$ at equally spaced time units. Finally, there are x batches of tiny jobs, each job of size $1/L^5$. For $i = 1, \ldots, x-1$, the i-th batch is released during $[(i-1)L^2 + L, iL^2]$ such that one job is released every $1/L^3$ units of time. The last batch of tiny jobs is released during $[(x-1)L^2, L^3]$, each job released every $1/L^3$ units of time.

Note that for each interval $[(i-1)L^2, (i-1)L^2 + L]$, the amount of work of medium jobs released in the interval is $(1-\epsilon(1-\epsilon/2)^i)L$. For each such interval, the LP solution places a $\epsilon(1-\epsilon/2)^i$ fraction of the big job at the beginning of this interval, and places $1-\epsilon(1-\epsilon/2)^i$ units of medium jobs in first come first served order every 1 time unit. Finally each tiny job is scheduled to an extent of 1 immediately upon release. It can be easily verified that each job is scheduled to an extent of at least 1 and hence this is a feasible LP solution. A simple calculation shows that the value of LP^* is at most $O(L^2)$ (the total flow time of big job is $\sum_i \epsilon(1-\epsilon/2)^i((i-1)L^2 + L) = O(L^2)$, similarly the penalty terms in the i-th batch contribute about $\epsilon(1-\epsilon/2)^i L^2/2$ and hence the total contribution is $O(L^2)$, the flow time of medium jobs is $O(L^2)$ and the tiny jobs is $O(L)$). We now claim that any non-preemptive schedule on a $(2-\epsilon)$-speed machine has flow time at least $\Omega(L^3)$. Clearly this is true if the big job is placed after L^3, and hence we assume that it is placed earlier. Another observation is that even if a single medium job (or even one time unit of big job) is placed during the intervals where tiny jobs arrive, then at least $\Omega(L^3)$ tiny jobs incur a flow time of at least 1/2. This implies that the big job must be placed entirely in the interval where medium jobs arrive. However, since the speed is only $(2-\epsilon)$, even if the large job is placed during the x-th batch (where the fewest number of medium jobs arrive), at least $L + L(1-\epsilon(1-\epsilon/2)^x) - L(2-\epsilon) \approx (\epsilon/2)L = \Omega(L)$ work cannot be completed within this batch and hence this will displace $\Omega(L)$ medium jobs, all of which need to move a distance of at least $\Omega(L^2)$ to avoid overlap with tiny jobs. Hence the total flow time of any schedule is $\Omega(L^3)$. As the number of jobs in the instance is $n = O(L^6)$, this implies an integrality gap of $\Omega(n^{1/6})$. ■

In contrast to the previous result, it is possible to obtain a positive result with speed $1+\epsilon$ processors using resource augmentation on the number of processors. More precisely, it is possible to obtain an $O(\frac{-\log\epsilon}{\epsilon})$-machine $1+O(\epsilon)$-speed $O(1)$-approximation polynomial-time algorithm. The jobs sizes are initially rounded down to the nearest integer power of $1+\epsilon$. We formulate a strong-time-indexed integer linear program. This integer linear program is not restricted to aligned schedules, but does incorporate the new lower bound on the flow of a job. The relaxed linear program is then solved. The classes of jobs are partitioned between each of the m processors. Jobs in class C_i are assigned to processor $i \bmod m$. If $m \geq (\log\frac{1}{\epsilon})/\log(1+\epsilon) = \Theta(-(\log\epsilon)/\epsilon)$ then the possible jobs sizes assigned to any one processor form a geometric sequence with ratio $1/\epsilon$. The pseudo-schedule is then created as before. The total size of the shorter jobs that overlap a bigger job in the pseudo-schedule can not be must longer than the size of the bigger job. So $1 + \Theta(\epsilon)$-speed resource augmentation is enough to change the pseudo-schedule into a feasible schedule.

5 Conclusions and Open Problems

We have given the first $O(1)$-speed $O(1)$-approximation algorithms for several fundamental problems. We believe that the techniques are general and should apply to other minsum scheduling objectives. We also believe that the constants can be made smaller, as we did not try to optimize the constants.

It would be very interesting to see if these techniques can be applied to multiple machines settings. A first step could be to obtain results for (the seemingly easier) throughput maximization on multiple machines. Note that since all the jobs need not be scheduled by the optimum, it is not clear how to extend the ideas of Chuzhoy *et al.* [5].

References

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 32–43, 1999.

[2] E. Balas. On the facial structure of scheduling polyhedra. *Mathematical Programming Study*, 24:179–218, 1985.

[3] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. S. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001.

[4] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM J. Comput.*, 31:332–352, 2001.

[5] J. Chuzhoy, S. Guha, S. Khanna, and S. Naor. Machine minimization for scheduling jobs with interval constraints. In *Proceedings of the 45th Annual Symposium on Foundations of Computer Science*, pages 81–90, 2004.

[6] J. Chuzhoy, R. Ostrovsky, and Y. Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, pages 348–356, 2001.

[7] M. R. Garey and D. S. Johnson. Two-processor scheduling with start-times and deadlines. *SIAM J. Comput.*, 1977.

[8] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, Aug. 1997.

[9] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

[10] B. Kalyanasundaram, K. Pruhs, and M. Velauthapillai. Scheduling broadcasts in wireless networks. *Journal of Scheduling*, 4(6):339 – 354, 2000.

[11] H. Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 418–426, May 1996.

[12] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32:163–200, 2001.

[13] C. A. Phillips, C. Stein, and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199–223, 1998.

[14] K. Pruhs, J. Sgall, and E. Torng. Online scheduling. In J. Leung, editor, *Handbook of Scheduling*. CRC Press, 2003.

[15] M. Queyranne and A. Schulz. Polyhedral approaches to machine scheduling. Technical Report Technical Report 474/1995, Technical University of Berlin, 1994.

[16] F. C. R. Spieksma. Approximating an interval scheduling problem. In *APPROX*, pages 169–180, 1998.