Scheduling with Outliers

Anupam Gupta* Ravishankar Krishnaswamy* Amit Kumar† Danny Segev ‡

Abstract

In classical scheduling problems, we are given jobs and machines, and have to schedule all the jobs to minimize some objective function. What if each job has a specified profit, and we are no longer required to process all jobs—we can schedule any subset of jobs whose total profit is at least a (hard) target profit requirement, while still approximately minimizing the objective function?

We refer to this class of problems as *scheduling with outliers*. This model was initiated by Charikar and Khuller (SODA'06) on the minimum max-response time in broadcast scheduling. In this paper, we consider three other well-studied scheduling objectives: the generalized assignment problem, average weighted completion time, and average flow time, and provide LP-based approximation algorithms for them. Our main results are:

- For the *minimum average flow time* problem on identical machines, we give a logarithmic approximation algorithm for the case of unit profits based on rounding an LP relaxation; we also show a matching integrality gap. While the LP relaxation has been used before, the rounding algorithm is a delicate one.
- For the *average weighted completion time* problem on unrelated machines, we give a constant-factor approximation. The algorithm is based on randomized rounding of the time-indexed LP relaxation strengthened by the knapsack-cover inequalities.
- For the *generalized assignment problem* with outliers, we give a simple reduction to GAP without outliers to obtain an algorithm whose makespan is within 3 times the optimum makespan, and whose cost is at most $(1 + \epsilon)$ times the optimal cost.

*Computer Science Department, Carnegie Mellon University. Supported in part by NSF awards CCF-0448095 and CCF-0729022, and an Alfred P. Sloan Fellowship.

†Department of Computer Science & Engineering, Indian Institute of Technology, Hauz Khas, New Delhi, India - 110016. Work partly done at MPI, Saarbrücken, Germany.

‡Sloan School of Management, Massachusetts Institute of Technology. Supported in part by NSF awards CCF-0448095 and CCF-0729022, and an Alfred P. Sloan Fellowship.

1 Introduction

In classical scheduling problems, we are given jobs and machines, and have to schedule all the jobs to minimize some objective function. *What if we are given a (hard) profit constraint, and merely want to schedule a "profitable" subset of jobs?* In this paper, we consider three widely studied scheduling objectives— makespan, weighted average completion time, and average flow-time—and give approximation algorithms for these objectives in this model of scheduling with outliers.

Formally, the *scheduling with outliers* model is as follows: given an instance of some classical scheduling problem, imagine each job j also comes with a certain *profit* π_j. Given a target profit Π, the goal is now to pick a subset of jobs S whose total profit $\sum_{j \in S} \pi_j$ is at least Π, and to schedule them to minimize the underlying objective function. (Equivalently, we could define the "budget" $B = \sum_j \pi_j - \Pi$, and discard a subset of "outlier" jobs whose total profit is at most B.) Note that this model introduces two different sources of computational difficulty: on one hand, the task of choosing a set of jobs to achieve the profit threshold captures the knapsack problem; on the other hand, the underlying scheduling problem may itself be an intractable problem.

The goal of picking some subset of jobs to process as efficiently as possible, so that we attain a minimum level of profit or "happiness", is a natural one. In fact, various problems of scheduling with job rejections have been studied previously: a common approach, studied by Bartal et al. [4], has been to study "prize-collecting" scheduling problems (see, e.g., [10, 3, 11, 18]), where we attempt to minimize the scheduling objective *plus the total profit of unscheduled jobs*. One drawback of this prize-collecting approach is that we lose fine-grained control on the individual quantities—the scheduling cost, and the lost profit—since we naïvely sum up these two essentially incomparable quantities. In fact, this makes our model (with a hard target constraint) interesting also from a technical standpoint: while we can reduce the prize-collecting problem to the target profit problem by guessing the lost profit in the optimal prize-collecting solution, reductions in the opposite direction are known only for a handful of problems with very restrictive structure (see Section 1.2 for a discussion).

To the best of our knowledge, the model we investigate was introduced by Charikar and Khuller [6], who considered the problem of minimizing the maximum response time in the context of broadcast scheduling; one of our results is to resolve an open problem from their paper. Scheduling problems with outliers were also implicitly raised in the context of model-based optimization with budgeted probes: Guha and Munagala [16] gave an LP-based algorithm for completion-time scheduling with outliers which violated budgets by a constant factor—we resolve an open problem in their paper by avoiding any violation of the budgets.

1.1 Our results

GAP and makespan. As a warm-up, we study the Generalized Assignment Problem, a generalization of the makespan minimization problem on unrelated machines, in Section 2. For this problem, we give a simple reduction to the non-outlier version of this problem to get a solution approximating the makespan and cost by factors of 3 and $(1+\epsilon)$ respectively. Recall that the best non-outlier guarantee is a 2-approximation [29] without violating the cost—however, it is easy to show that in the presence of outliers the $(1+\epsilon)$ loss in cost are unavoidable unless P = NP.

Average completion time. We then consider the problem of minimizing the sum of weighted completion times on unrelated machines with release dates in Section 3.

Theorem 1.1. *For $R|r_j, \text{outliers}|\sum_j w_j C_j$, there is a randomized $O(1)$-approximation algorithm.*

Our algorithm is based on approximately solving the time-indexed LP relaxation of Schulz and Skutella [27] strengthened with *knapsack-cover* inequalities followed by randomized rounding. We improve on this

result to obtain an FPTAS for *unweighted* sum of completion times on a constant number of machines. (The best non-outlier upper bound for $R|r_j|\sum_j w_j C_j$ is a 2-approximation due to Skutella [30]; this problem is also known to be APX-hard [22].)

Average flow time. This is the technical heart of the paper. The problem is to minimize the average (preemptive) flow time on identical machines $P|r_j, pmtn, \text{outliers}|\sum F_j$. Our main result is:

Theorem 1.2. *For $P|r_j, pmtn, \text{outliers}|\sum F_j$, when all jobs have unit profits, there is an $O(\log P)$-approximation algorithm, where P is the ratio between the largest and smallest processing times.*

This comes close to matching the best known result of $O(\log \min\{P, n/m\})$ for the non-outlier version due to Leonardi and Raz [23]. However, this problem seems to be much harder with outliers, as we get the same approximation even on a single machine, in contrast to the non-outlier single-machine case (which can be solved optimally). We show our approach is tight, as the LP relaxation we use has an $\Omega(\log P)$ integrality gap.

The algorithm rounds a linear-programming relaxation originally suggested in [13]; however, we need new ideas for the rounding algorithm over those used by [13]. At a high-level, here is the idea behind our rounding algorithm: the LP might have scheduled each job to a certain fractional amount, and hence we try to swap "mass" between jobs of near-equal processing times in order to integrally schedule a profitable subset of jobs. However, this swapping operation is a delicate one, and merely swapping mass locally between nearby jobs has a bad algorithmic gap. Furthermore, we need to handle jobs that are only approximately equal in size, which leads to additional difficulties. (For a more detailed high-level sketch of these issues, please read Section 4.2.)

1.2 Related work

Scheduling with rejections. As mentioned above, previous papers on this topic considered the "prize-collecting" version which minimizes the scheduling objective plus the total profit of unscheduled jobs; their techniques do not seem to extend to scheduling with outliers, in which we have a strict budget on the total penalty of rejected jobs. Bartal et al. [4] considered offline and online makespan minimization and gave best-possible algorithms for both cases. Makespan minimization with preemptions was investigated by [18, 28]. Epstein et al. [11] examined scheduling unit-length jobs. Engels et al. [10] studied the prize-collecting version of weighted completion-time minimization (on single or parallel machines), and gave PTASs or constant-factor approximations for these problems; they also proposed a general framework for designing algorithms for such problems.

Outlier versions of other problems. Also called *partial-covering* problems, these have been widely studied: e.g., the k-MST problem [12], the k-center and facility location problem [7] and the k-median problem with outliers [8], partial vertex cover (e.g., [25] and references therein) and k-multicut [15, 24]. Chudak et al. [9] distilled ideas of Jain and Vazirani [19] on converting "Lagrange-multiplier preserving" algorithms for prize-collecting Steiner tree into one for k-MST; Könemann et al. [21] gave a general framework to convert prize-collecting algorithms into algorithms for outlier versions (see also [26]). We cannot use these results, since it is not clear how to make the algorithms for prize-collecting scheduling problems to also be Langrange-multiplier preserving, or whether the above-mentioned framework is applicable in scheduling-related scenarios.

2 GAP and Makespan

As a warm-up, we consider the generalized assignment problem, which is an extension of minimizing makespan on unrelated machines with outliers. Formally, the instance $\mathcal{I}$ has m machines and n jobs. Each job j has a processing time of p_{ij} on machine i, an assignment cost of c_{ij}, and a profit of π_j. Given

a profit requirement Π, cost bound C and makespan bound T, the goal is to obtain a feasible schedule satisfying these requirements (or to declare infeasibility). Of course, since the problem is NP-hard, we look at finding solutions where we violate the cost and makespan bounds, but not the (hard) profit requirement. We now show how to reduce this problem to the non-outlier version studied earlier, while incurring small additional losses in the approximation guarantees.

Theorem 2.1. *Given an instance $\mathcal{I}$ of GAP-with-outliers with optimal cost C, and makespan T, there is a polynomial time algorithm to output an assignment with cost $(1+\epsilon)C$ and makespan $3T$.*

Proof. Given the instance $\mathcal{I}$, construct the following instance $\mathcal{I}'$ of the standard GAP (where there are no profits or outliers). There are $m+1$ machines: machines $1, 2, \ldots, m$ are the same as those in $\mathcal{I}$, while machine $m+1$ is a "virtual profit machine". We have n jobs, where job j has a processing time of p_{ij} and an assignment cost of c_{ij} when scheduled on machine i (for $1 \leq i \leq m$). If job j is scheduled on the virtual machine $m+1$, it incurs a processing time of π_j and cost zero: i.e., $p_{(m+1)j} = \pi_j$ and $c_{(m+1)j} = 0$. For this instance $\mathcal{I}'$, we set a cost bound of C, makespan bound of T for all machines $1 \leq i \leq m$, and a makespan bound of $T_{vpm} := (\sum_{j=1}^{n} \pi_j) - \Pi$ for the virtual profit machine. Note that any feasible solution for $\mathcal{I}$ is also feasible for $\mathcal{I}'$, with the outliers being scheduled on the virtual profit machine, since the total profit of the outliers is at most $T_{vpm} = (\sum_{j=1}^{n} \pi_j) - \Pi$.

We can now use the algorithm of Shmoys and Tardos [29] which guarantees an assignment $\mathbf{S}$ for the GAP instance $\mathcal{I}'$ with the following properties: *(a)* The cost of assignment $\mathbf{S}$ is at most C, *(b)* the makespan induced by $\mathbf{S}$ on machine i (for $1 \leq i \leq m$) is at most $T + \min\{\max_j p_{ij}, T\}$, and *(c)* The makespan of $\mathbf{S}$ on the virtual machine $m+1$ is at most $((\sum_{i=1}^{n} \pi_j) - \Pi) + \max_j \pi_j$.

Note that this assignment $\mathbf{S}$ is *almost* feasible for the outlier problem $\mathcal{I}$—the makespan on any real machine is at most $T + \max_j p_{ij}$, the assignment cost is at most C—however, the profit of the scheduled jobs is only guaranteed to be at least $\Pi - \max_j \pi_j$. But it is easy to fix this shortcoming: we choose a job j' assigned by $\mathbf{S}$ to the virtual machine which has the largest profit, and schedule j' on the machine where it has the least processing time. Now the modified assignment has cost at most $C + \max_{ij'} c_{ij'}$, makespan at most $T + 2\min\{\max_j p_{ij}, T\}$, and the total profit of the scheduled jobs is at least Π. (We assume that any job j where $\min_i p_{ij} > T$ has already been discarded.) This is almost what we want, apart from the cost guarantee. So suppose we "guess" the $1/\epsilon$ most expensive assignments in OPT (in time $O(mn^{1/\epsilon})$), and hence we can focus only on the jobs having $c_{ij} \leq \epsilon C$ for all possible remaining assignments. Now the cost of the assignment is $C + \max_{ij'} c_{ij'} \leq C(1+\epsilon)$, and the makespan is at most $3T$. This completes the proof. ∎

In fact, the $(1+\epsilon)$ loss in cost is inevitable since we can reduce the knapsack problem to the single machine makespan minimization with outliers problem: values of items become profits of jobs, and their weights become the assignment cost; the weight budget is the cost budget, and the required value is the required profit. As for the makespan guarantee, the 3/2-hardness of Lenstra et al. [22] carries over.

3 Weighted Sum of Completion Times

We now turn our attention to average completion time—in particular, to $R|r_j, \text{outliers}|\sum_j w_j C_j$. The main result of this section is a constant factor approximation for this problem. Not surprisingly, the integrality gap of standard LP relaxations is large[1], and hence we strengthen the time-indexed formulation with the so-called knapsack-cover inequalities [5, 31]. We show that a randomized rounding scheme similar to that of Schulz and Skutella [27] gives us the claimed guarantees on the objective function, and while preserving the profit requirements with constant probability.

[1]Implicit in the work of Guha and Munagala [16] is an algorithm which violates the profit requirement by a constant factor; they also comment on the integrality gap, and pose the problem of avoiding this violation.

3.1 A Constant Approximation for Weighted Sum of Completion Times

We have a collection of m machines and n jobs, where each job j is associated with a profit π_j, a weight w_j, and a release date r_j. When job j is scheduled on machine i, it incurs a processing time of p_{ij}. Given a parameter $\Pi > 0$, the objective is to identify a set of jobs S and a feasible schedule such that $\sum_{j \in S} \pi_j \geq \Pi$ and such that $\sum_{j \in S} w_j C_j$ is minimized. Here, C_j denotes the completion time of job j.

3.1.1 A Time Indexed LP Relaxation

For the non-outlier version, in which all jobs have to be scheduled, Schulz and Skutella [27] gave a constant factor approximation by making use of a time-indexed LP. We first describe a natural extension of their linear program to the outlier case, while also *strengthening* it.

$$
\begin{array}{lll}
\text{minimize} & \sum_{j=1}^{n} w_j C_j & \\
\text{subject to} & (1) \quad C_j = \sum_{i=1}^{m} \sum_{t=0}^{T} \left(\frac{x_{ijt}}{p_{ij}} \left(t + \frac{1}{2}\right) + \frac{x_{ijt}}{2} \right) & \forall\, j \\
& (2) \quad y_j = \sum_{i=1}^{m} \sum_{t=0}^{T} \frac{x_{ijt}}{p_{ij}} & \forall\, j \\
& (3) \quad \sum_{j=1}^{n} x_{ijt} \leq 1 & \forall\, i, t \\
& (4) \quad \sum_{j \notin \mathcal{A}} \pi_j^{\mathcal{A}} y_j \geq \Pi - \Pi(\mathcal{A}) & \forall\, \mathcal{A} : \Pi(\mathcal{A}) < \Pi \\
& (5) \quad x_{ijt} = 0 & \forall\, i, j, t : t < r_j \\
& (6) \quad x_{ijt} \geq 0,\ 0 \leq y_j \leq 1 & \forall\, i, j, t
\end{array}
$$

In this formulation, the variable x_{ijt} stands for the fractional amount of time machine i spends on processing job j in the time interval $[t, t+1)$; note that the LP schedule may be preemptive. The variable C_j, defined by constraint (1), is a measure for the completion time of job j. In any integral solution, where job j is scheduled from t to $t + p_{ij}$ on a single machine i, it is not difficult to verify that C_j evaluates to $t + p_{ij}$. The variable y_j, defined by constraint (2), is the fraction of job j being scheduled. Constraint (3) ensures that machine i spends at most one unit of processing time in $[t, t+1)$. Constraints (5) and (6) are additional feasibility checks.

We first observe that replacing the set of constraints (4) by a single inequality of the form $\sum_{j=1}^{n} \pi_j y_j \geq \Pi$ would result in an unbounded integrality gap – consider a single job of profit M, and $\Pi = 1$; the LP can schedule a $1/M$ fraction of the job, incurring a cost which is only $1/M$ times the optimum. We therefore add in the family of constraints (4), known as the *knapsack-cover (KC) inequalities*. Let $\mathcal{A}$ be any set of jobs, and let $\Pi(\mathcal{A}) = \sum_{j \in \mathcal{A}} \pi_j$ be the sum of profits over all jobs in $\mathcal{A}$. Then, $[\Pi - \Pi(\mathcal{A})]^+$ is the profit that needs to be collected by jobs not in $\mathcal{A}$ when all jobs in $\mathcal{A}$ are scheduled. Further, if $\mathcal{A}$ does not fully satisfy the profit requirement, any job $j \notin \mathcal{A}$ has a marginal contribution of at most $\pi_j^{\mathcal{A}} = \min\{\pi_j, \Pi - \Pi(\mathcal{A})\}$. Therefore, for every set $\mathcal{A}$ such that $\Pi(\mathcal{A}) < \Pi$, we add a constraint of the form $\sum_{j \notin \mathcal{A}} \pi_j^{\mathcal{A}} y_j \geq \Pi - \Pi(\mathcal{A})$. Note that there are exponentially many such constraints, and hence we cannot naively solve this LP.

"Solving" the LP. We will not look to find an optimal solution to the above LP; for our purposes, it suffices to compute a solution vector $(\widehat{x}, \widehat{y}, \widehat{C})$ satisfying the following:

(a) Constraints (1)-(3) and (5)-(6) are satisfied.
(b) Constraint (4) is satisfied for the single set $\{j : \widehat{y}_j \geq 1/2\}$.
(c) $\sum_{j=1}^{n} w_j \widehat{C}_j \leq 2 \cdot \mathsf{Opt}$, where Opt denotes the cost of an optimal integral solution.

We compute this solution vector by first guessing Opt up to a multiplicative factor of 2 (call the guess $\widetilde{\mathsf{Opt}}$), and add to the LP the explicit constraint $\sum_{j=1}^{n} w_j C_j \leq \widetilde{\mathsf{Opt}}$. Then, we solve the LP using the ellipsoid algorithm. For the separation oracle, in each iteration, we check if the current solution satisfies properties (a)-(c) above. If none of these properties is violated, we are done; otherwise, we have a

violated constraint. We now present our rounding algorithm (in Algorithm 1), based on the non-outlier algorithm of [27].

Algorithm 1 Weighted Sum of Completion Times

1: **given** a solution vector $(\widehat{x}, \widehat{y}, \widehat{C})$ satisfying properties (a)-(c), **let** $\mathcal{A}^*$ be the set $\{j \,|\, \widehat{y}_j \geq 1/2\}$.
2: **for** each job j, do the following steps
  2a: **if** $j \in \mathcal{A}^*$, for each (i,t) pair, set $l_{ijt} = \widehat{x}_{ijt}/(p_{ij}\widehat{y}_j)$. Note that for such jobs $j \in \mathcal{A}^*$, we have $\sum_{i=1}^{m}\sum_{t=0}^{T} l_{ijt} = 1$ from constraint (2) of the LP.
  2b: **if** $j \notin \mathcal{A}^*$, set $l_{ijt} = 2\widehat{x}_{ijt}/p_{ij}$. In this case, note that $\sum_{i=1}^{m}\sum_{t=0}^{T} l_{ijt} = 2\widehat{y}_j$.
  2c: **partition** the interval $[0,1]$ in the following way: assign each (i,t) pair a sub-interval I_{it} of $[0,1]$ of length l_{ijt} such that these sub-intervals are pairwise disjoint. Then choose a uniformly random number $r \in [0,1]$ and set τ_j to be the (i,t) pair s.t $r \in I_{it}$. If there is no such (i,t) pair, leave j unmarked.
3: **for** each machine i, consider the jobs such that $\tau_j = (i,*)$; order them in increasing order of their marked times; schedule them as early as possible (subject to the release dates) in this order.

3.2 Analysis

We now show that the expected weighted sum of completion times is $O(1)\mathsf{Opt}$, and also that with constant probability, the total profit of the jobs scheduled is at least Π.

Lemma 3.1. *The expected weighted sum of completion times is at most* $16 \cdot \mathsf{Opt}$.

Proof. Let C_j^R be a random variable, standing for the completion time of job j; if this job has not been scheduled, we set $C_j^R = 0$. Since $\sum_{j=1}^{n} w_j \widehat{C}_j \leq 2 \cdot \mathsf{Opt}$, it is sufficient to prove that $\mathrm{E}[C_j^R] \leq 8\widehat{C}_j$ for every j. To this end, note that

$$\mathrm{E}\left[C_j^R\right] = \sum_{i=1}^{m}\sum_{t=0}^{T} \Pr[\tau_j = (i,t)] \cdot \mathrm{E}\left[C_j^R | \tau_j = (i,t)\right] \leq \sum_{i=1}^{m}\sum_{t=0}^{T} \frac{2\widehat{x}_{ijt}}{p_{ij}} \cdot \mathrm{E}\left[C_j^R | \tau_j = (i,t)\right] \ ,$$

where the last inequality holds since $\Pr[\tau_j = (i,t)] = l_{ijt} \leq 2\widehat{x}_{ijt}/p_{ij}$, regardless of whether $j \in \mathcal{A}^*$ or not. Now let us upper bound $\mathrm{E}[C_j^R | \tau_j = (i,t)]$. The total time for which job j must wait before being processed on machine i can be split in the worst case into: (a) the idle time on this machine before j is processed, and (b) the total processing time of other jobs marked (i,t') with $t' \leq t$. If job j has been marked (i,t), the idle time on machine i before j is processed is at most t. In addition, the total expected processing time mentioned in item (b) is at most

$$\sum_{k \neq j} p_{ik} \sum_{t'=0}^{t} \Pr\left[\tau_k = (i,t') \,|\, \tau_j = (i,t)\right] = \sum_{k \neq j} p_{ik} \sum_{t'=0}^{t} \Pr\left[\tau_k = (i,t')\right]$$
$$\leq \sum_{k \neq j} p_{ik} \sum_{t'=0}^{t} \frac{2\widehat{x}_{ikt'}}{p_{ik}} = 2\sum_{t'=0}^{t}\sum_{k \neq j} \widehat{x}_{ikt'} \leq 2(t+1) \ ,$$

where the last inequality follows from constraint (3). Combining these observations and constraint (1), we have

$$\mathrm{E}\left[C_j^R\right] \leq 2\sum_{i=1}^{m}\sum_{t=0}^{T} \frac{\widehat{x}_{ijt}}{p_{ij}} \left(t + 2(t+1) + p_{ij}\right) \leq 8\sum_{i=1}^{m}\sum_{t=0}^{T} \left(\frac{\widehat{x}_{ijt}}{p_{ij}}\left(t + \frac{1}{2}\right) + \frac{\widehat{x}_{ijt}}{2}\right) = 8\widehat{C}_j \ .$$

■

Lemma 3.2. *The randomized rounding algorithm produces a schedule that meets the profit constraint with probability at least* $1/5$.

Proof. Clearly, when the jobs in $\mathcal{A}^*$ collectively satisfy the profit requirement, we are done since the algorithm picks every job in this set. In the opposite case, consider the Knapsack Cover inequality for $\mathcal{A}^*$, stating that $\sum_{j \notin \mathcal{A}^*} \pi_j^{\mathcal{A}^*} \widehat{y}_j \geq \Pi - \Pi(\mathcal{A}^*)$. The total profit collected from these jobs can be lower bounded by $Z = \sum_{j \notin \mathcal{A}^*} \pi_j^{\mathcal{A}^*} Z_j$, where Z_j is a random variable indicating whether job j is picked.

Since our rounding algorithm picks all jobs in $\mathcal{A}^*$, the profit requirement is met if Z is at least $\Pi - \Pi(\mathcal{A}^*)$. To provide an upper bound on the probability that Z falls below $\Pi - \Pi(\mathcal{A}^*)$, notice that by the way the algorithm marks jobs in Step 2, we have that each job not in $\mathcal{A}^*$ is marked with probability $2\widehat{y}_j$, *independently* of the other jobs. Therefore,

$$\mathrm{E}[Z] = \mathrm{E}\big[\textstyle\sum_{j \notin \mathcal{A}^*} \pi_j^{\mathcal{A}^*} Z_j\big] = 2 \textstyle\sum_{j \notin \mathcal{A}^*} \pi_j^{\mathcal{A}^*} \widehat{y}_j \geq 2(\Pi - \Pi(\mathcal{A}^*)) \ .$$

Consequently, if we define $\alpha_j = \pi_j^{\mathcal{A}^*}/(\Pi - \Pi(\mathcal{A}^*))$, then

$$\begin{aligned} \Pr[Z \leq \Pi - \Pi(\mathcal{A}^*)] &= \Pr\left[\textstyle\sum_{j \notin \mathcal{A}^*} \frac{\pi_j^{\mathcal{A}^*}}{\Pi - \Pi(\mathcal{A}^*)} Z_j \leq 1\right] \leq \Pr\left[\textstyle\sum_{j \notin \mathcal{A}^*} \alpha_j Z_j \leq \frac{\mathrm{E}[\sum_{j \notin \mathcal{A}^*} \alpha_j Z_j]}{2}\right] \\ &\leq \exp\big(-\tfrac{1}{8} \cdot \mathrm{E}\big[\textstyle\sum_{j \notin \mathcal{A}^*} \alpha_j Z_j\big]\big) \leq e^{-1/4} < \tfrac{4}{5} \ , \end{aligned}$$

where the first and third inequalities hold since $\mathrm{E}[\sum_{j \notin \mathcal{A}^*} \alpha_j Z_j] \geq 2$, and the second inequality follows from bounding the lower tail of the sum of independent $[0,1]$ r.v.s (see, e.g., [1, Thm. 3.5]). ∎

The above two lemmas combine to give the following theorem.

Theorem 3.3. *For $R|r_j, \text{outliers}|\sum_j w_j C_j$, there is a randomized $O(1)$-approximation algorithm.*

While the LP formulation as stated has exponentially many time intervals of length 1, we can make our algorithm fully polynomial in the size of the input (with a small loss in approximation guarantee) by considering geometrically increasing sizes [17] for the time intervals.

In Appendix B.1, we show that given K *different* profit requirements, our algorithm can be modified to give an $O(\log K)$-approximation.

3.3 Single Machine, Identical Weights

In this section, we show how we can get an FPTAS using dynamic programming for the problem of minimizing the unweighted sum of completion times on a constant number of machines. For simplicity, we first give the complete proof for the case of a single machine, and sketch how to extend it for a constant number of machines.

Single Machine, Identical Weights. We are given a collection of n jobs where job j is associated with a processing time p_j and a profit π_j. Given a target profit of $\Pi > 0$, the goal is to identify a set of jobs S and a corresponding single-machine schedule such that $\sum_{j \in S} \pi_j \geq \Pi$ and $\sum_{j \in S} C_j$ is minimized (where C_j is the completion time of job j).

Dynamic program. Suppose $p_1, \ldots, p_n$ are integers such that $p_1 \leq \cdots \leq p_n$. Let $\mathsf{profit}(j, C, L)$ be the maximum profit that can be collected by scheduling a subset of jobs $\{1, \ldots, j\}$ such that their sum of completion times is *at most* C and makespan is *exactly* L. Then, the following recurrence holds:

$$\mathsf{profit}(j, C, L) = \max\{\mathsf{profit}(j-1, C, L), \pi_j + \mathsf{profit}(j-1, C-L, L-p_j)\}$$

To better understand the above equation, notice that if job j is picked by an optimal schedule, it will not be scheduled before any of the jobs $\{1, \ldots, j-1\}$ since the *shortest processing time* strategy is optimal for a fixed set of jobs ([20]). Therefore, consider a set of jobs $\{1, \ldots, j\}$ that have a bound C on their sum of completion times and let L be their makespan. If j is scheduled, the jobs $\{1, \ldots, j-1\}$ *must* have a residual makespan of $L - p_j$ and a bound of $C - L$ on the sum of completion times since job j

incurs a completion time of L by virtue of it being scheduled last among $\{1, \ldots, j\}$; we also collect a profit of π_j in this case. On the other hand, if j is not scheduled, C and L remain the same but we don't collect any profit. Now, given this recurrence, the goal is to find the minimal C and some L such that $\mathsf{profit}(n, C, L) \geq \Pi$. This can be solved by dynamic programming, with running time $O(nC_{max}L_{max})$. Since $C \leq n^2 p_n$ and $L \leq np_n$, the running time is $O(n^4 p_n^2)$, i.e. pseudo-polynomial.

Therefore, the above dynamic program can be used to compute an optimal solution in polynomial time when all processing times are small integers. We now apply scaling techniques to obtain an FPTAS to handle arbitrary processing times.

Handling general instances. Given an instance $\mathcal{I}$ of the original problem, we begin by "guessing" $P_{\max}$, the maximum processing time of a job that is scheduled in some fixed optimal solution. We now create a new instance $\mathcal{I}'$ in which every job j with $p_j > P_{\max}$ is discarded; other jobs get a scaled processing time of $p'_j = \lceil p_j/K \rceil$, where $K = (2\epsilon P_{\max})/(n(n+1))$. Notice that the scaled processing times of remaining jobs are integers in $[0, \lceil n(n+1)/(2\epsilon) \rceil]$. We can therefore find in $O(n^8/\epsilon^2)$ time an optimal subset of jobs $\mathcal{J}_{\mathcal{I}'}$ to be scheduled in $\mathcal{I}'$, and return this set as a solution for $\mathcal{I}$.

Theorem 3.4. *Scheduling the jobs $\mathcal{J}_{\mathcal{I}'}$ in order of non-decreasing processing times guarantees that their sum of completion times is at most $(1+\epsilon)\mathsf{Opt}(\mathcal{I})$.*

Proof. We begin by relating $\mathsf{Opt}(\mathcal{I})$ to $\mathsf{Opt}(\mathcal{I}')$. For this purpose, suppose that $\mathcal{J}_{\mathcal{I}} = \{j_1, \ldots, j_R\}$ in an optimal solution to $\mathcal{I}$. Then,

$$\mathsf{Opt}(\mathcal{I}') \leq \sum_{r=1}^{R} \sum_{s=1}^{r} p'_{j_s} \leq \sum_{r=1}^{R} \sum_{s=1}^{r} \left(\frac{p_{j_s}}{K} + 1 \right) \leq \frac{\mathsf{Opt}(\mathcal{I})}{K} + \frac{n(n+1)}{2} \ .$$

Now suppose that $\mathcal{J}_{\mathcal{I}'} = \{j'_1, \ldots, j'_Q\}$. Then, the sum of completion times that results from scheduling $j'_1, \ldots, j'_Q$ in this exact order is

$$\begin{aligned} \sum_{r=1}^{Q} \sum_{s=1}^{r} p_{j'_s} &\leq K \sum_{r=1}^{Q} \sum_{s=1}^{r} p'_{j'_s} \\ &= K \cdot \mathsf{Opt}(\mathcal{I}') \\ &\leq \mathsf{Opt}(\mathcal{I}) + \frac{n(n+1)K}{2} \\ &= \mathsf{Opt}(\mathcal{I}) + \epsilon P_{\max} \\ &\leq (1+\epsilon)\mathsf{Opt}(\mathcal{I}) \ . \end{aligned}$$

where the last inequality holds since $P_{\max}$ is a lower bound on $\mathsf{Opt}(\mathcal{I})$. ■

Constant number of machines.

We finally consider the case when there is a constant number of identical machines, say m. To this end, let $\mathsf{profit}(j, C, L_1, L_2, \ldots, L_m)$ be the maximum profit that can be collected by scheduling a subset of the jobs $1, \ldots, j$ such that their sum of completion times is at most C and such that the makespan is exactly L_i on machine i. Then $\mathsf{profit}(j, C, L_1, \ldots, L_m)$ can be written as

$$\max \left\{ \mathsf{profit}(j-1, C, L_1, \ldots, L_m), \max_i \left(\pi_j + \mathsf{profit}(j-1, C - L_i, L_1, \ldots, L_i - p_j, \ldots, L_m) \right) \right\} \ .$$

When $m = O(1)$, the size of this dynamic program is still polynomial in n. The remaining analysis is similar to the one for a single machine.

4 Minimizing Average Flow Time on Identical Machines

Finally, we consider the problem of minimizing the average (preemptive) flow time on identical machines $(P|r_j, pmtn, \text{outliers}|\sum F_j)$ with *unit profits*. We present an LP rounding algorithm that produces a preemptive non-migratory (no job is scheduled on multiple machines) schedule whose flow time is within $O(\log P)$ of the optimal, where P is the ratio between the largest and smallest processing times.

This is the technical heart of the paper; in sharp contrast to the problems in the previous two sections, it is not clear how to easily change the existing algorithms for this problem to handle the outliers case—while we use the same LP as in previous works, our LP rounding algorithm for the outlier case has to substantially extend the previous non-outlier rounding algorithm. Since our algorithms are somewhat involved, we first present the algorithm for a *single machine*, and subsequently sketch how to extend it to multiple identical machines. For the rest of this section, consider the following setup: we are given a single machine and a collection of n jobs where each job j has a release date $r_j \in \mathbb{Z}$ and a processing time $p_j \in \mathbb{Z}$. Given a parameter $\Pi > 0$, we want to identify a set of jobs S and a preemptive schedule minimizing $\sum_{j \in S} F_j$ (where $F_j = C_j - r_j$) subject to $|S| \geq \Pi$.

4.1 The Flow-time LP Relaxation and an Integrality Gap

Our LP relaxation is a natural outlier extension of one used in earlier flow-time algorithms ([13, 14]). We first describe what the variables and constraints correspond to: *(i)* f_j is the fractional flow time of job j, *(ii)* x_{jt} is the fraction of job j scheduled in the time interval $[t, t+1)$, and *(iii)* y_j is the fraction of job j scheduled. Constraint (1) keeps track of the flow time of each job, while constraints (2), (3), and (4) are to make sure the solution is feasible with respect to the profit constraint. Notice that in constraint (1), we use the quantity $\widetilde{p}_j$ (which denotes the processing time p_j rounded up to the next power of 2), instead of p_j. Also, this modification is present only in constraint (1) which dictates the LP cost, and not in constraint (2) which measures the extent to which each job is scheduled. The quantity T is a guess for the time at which the optimal solution completes processing jobs (in fact, any upper bound of it would suffice). We also assume that a parameter $k^* \in \mathbb{Z}$ was guessed in advance, such that the optimal solution only schedules jobs with $p_j \leq 2^{k^*}$. Our algorithm would have running time which is polynomial in T and n.

$$
\begin{array}{lll}
\text{minimize} & \sum_{j=1}^{n} f_j & \\
\text{subject to} & (1) \quad f_j = \displaystyle\sum_{t=0}^{T} \left(\frac{x_{jt}}{\widetilde{p}_j} \left(t + \frac{1}{2} - r_j \right) + \frac{x_{jt}}{2} \right) & \forall\, j \\
& (2) \quad p_j y_j = \sum_{t=0}^{T} x_{jt} & \forall\, j \\
& (3) \quad \sum_{j=1}^{n} x_{jt} \leq 1 & \forall\, t \\
& (4) \quad \sum_{j=1}^{n} y_j \geq \Pi & \\
& (5) \quad x_{jt} = 0 & \forall\, j, t : t < r_j \\
& (6) \quad x_{jt} \geq 0,\ 0 \leq y_j \leq 1 & \forall\, j, t
\end{array}
$$

Given the above LP, we first claim that it is indeed a *relaxation*.

Lemma 4.1 (Relaxation)**.** $\mathsf{Opt}(\mathsf{LP}) \leq \mathsf{Opt}$, *where* Opt *denotes the optimal sum of flow times.*

Proof. Given an optimal solution for the given instance, we construct a corresponding LP solution in a natural way, by setting $x_{jt} = \Delta p_j$ when the optimal solution schedules a Δ fraction of job j in the time interval $[t, t+1)$. It is easy to verify that the profit constraint is satisfied. Now, consider a particular job j scheduled in the optimal solution. We proceed by showing that the term $f'_j = \sum_{t=0}^{T} (\frac{x_{jt}}{p_j}(t + \frac{1}{2} - r_j) + \frac{x_{jt}}{2})$ is a lower bound on the flow time of j (notice that f'_j has p_j in the denominator where f_j had $\widetilde{p}_j$).

Suppose the optimal solution completes processing j at C_j. The flow time is therefore $C_j - r_j$, whereas the worst case for the LP is when j is contiguously scheduled in the time interval $[C_j - p_j, C_j)$; otherwise, some fraction is scheduled earlier, and the contribution to f'_j can only decrease. Consequently,

$$f'_j \leq \sum_{t=C_j-p_j}^{C_j-1} \frac{t+1/2-r_j}{p_j} + \frac{p_j}{2} = C_j - r_j \ .$$

Since $f_j \leq f'_j$, we have $f_j \leq C_j - r_j$. The lemma follows by summing over all jobs scheduled by OPT. ■

Before getting into the details of our algorithm, to gain more intuition for this relaxation, we demonstrate that it has an integrality gap of $\Omega(\log P)$, where P is the ratio between the largest and smallest processing times in an optimal solution.

Theorem 4.2 (Integrality Gap)**.** *There are instances in which* $\mathsf{Opt} = \Omega(\log P) \cdot \mathsf{Opt}(\mathrm{LP})$.

Proof. Consider an instance where there are $k+1$ *large* jobs numbered $1, 2, \ldots, k+1$. Jobs $1, 2, \ldots, k$ have processing times $2^2, 2^3, \ldots, 2^{k+1}$ respectively and job $k+1$ has a processing time of 2^{k+1}. In addition, there are $M = M(k)$ *small* jobs of unit processing time, where M is a parameter whose value will be determined later. Large jobs $1, 2, \ldots, k$ arrive in decreasing order of processing time, where job j arrives at the beginning of the white block numbered j in Figure 1. White block j occupies 2^j time units. Job $k+1$ arrives at the beginning at the white block numbered $k+1$ which occupies 2^{k+1} time units. There is also a large grey block occupying M time units; the arrivals of small jobs are uniformly spaced in this block (starting at the left endpoint) with a gap of 1. Now suppose we are required to schedule $M + k/2 + 1$ jobs.

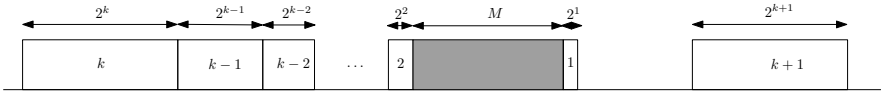

Figure 1: A schematic description of the integrality gap instance.

We first observe that the optimal schedule picks every small job (as well as $k/2+1$ large jobs). To see why, suppose one or more small jobs have not been picked, let q be the minimal index of such a job. Note that once we pick a subset of $M + k/2 + 1$ jobs, an optimal schedule is determined by employing the *shortest remaining processing time* rule (see, for example, [2]). Further, from the sizes of the white blocks, we see that even if a large job is scheduled without being preempted since its release date, it would have a remaining processing time of at least 2 at the beginning of the grey block. Therefore, from the SRPT rule, it is clear that the first $q-1$ small jobs are scheduled in the first $q-1$ time units of the grey block. Thus, at the point when job q is released, any large job has a remaining processing time of at least 2. It follows that, by picking q and dropping some large job, we can obtain a smaller flow time, implying that the schedule under consideration cannot be optimal.

Further, it is optimal to schedule large job $k+1$. If there is a solution which does not, we can schedule it while skipping one additional large job out of jobs $1, 2, \ldots, k$ to improve on the average flow time (this holds if $M \geq 2^{k+1}$, which we will ensure later). Therefore, the value of k^* in the LP (which denotes the largest class scheduled in an optimal solution) will be $k+1$.

Based on the above SRPT observation, we can conclude that each small job will be contiguously processed to completion immediately after its release date, and as a result no large job (from the set of jobs $1, 2, \ldots, k$) can be completed any sooner than the right endpoint of the grey block. Hence, every large job picked incurs a flow time of at least M, meaning that $\mathsf{Opt} \geq Mk/2$. On the other hand, a fractional solution can fully schedule *every* small job as soon as it arrives, and schedule *half* of each

large job j into white block j. It can also schedule large job $k+1$ completely into white block $k+1$. It is not difficult to verify that the cost of this solution is at most $M + \sum_{j=1}^{k} 2^j + 2^{k+1} < M + 2^{k+2}$. Therefore, by setting $M = 2^{k+1}$, we have

$$\frac{\mathsf{Opt}}{\mathsf{Opt}(\mathrm{LP})} \geq \frac{Mk}{2(M+2^{k+2})} = \frac{2^{k+1}k}{2(2^{k+1}+2^{k+2})} = \frac{k}{6} = \Omega(\log P) \ ,$$

where the last equality holds since $P = 2^{k+1}$. ∎

Note that this gap instance is on a single machine, for which we know that the shortest remaining processing time policy (SRPT) is optimal in the non-outlier case. However our results eventually show that this is as bad as it gets—we show an upper bound of $O(\log P)$ for the integrality gap even for identical machines!

4.2 The Flow-time Rounding Algorithm: General Game Plan and Some Hurdles

Before we present our algorithm in detail, let us give a high-level picture and indicate some of the complicating factors over the earlier work. Previous LP-based rounding techniques [13, 14] relied on the fact that if we rearrange the jobs of length roughly 2^k—call such jobs "class-k" jobs—among the time slots they occupy in the fractional solution, the objective function does not change much; these algorithms then use this rearrangement to make the schedule feasible (no job simultaneously scheduled on two machines) and even non-migratory across machines. We are currently considering the single machine case, so these issues are irrelevant for the time being (and we will come back to them later)—however, we need to handle jobs that are fractionally picked by the LP. In particular, we need to swap "mass" between jobs to pick an integral number of jobs to schedule. And it is this step which increases the LP cost even in the case of a single machine. Note that we essentially care only about the y_j value for each job j, which indicates the extent to which this job is scheduled—if we could make them integral without altering the objective by much, we would be done!

However, naïve approaches to make the y_j's integral may have bad approximation guarantees. E.g., consider taking two consecutive fractional jobs j and j' with similar processing times (observe that jobs with similar processing times have similar contributions to the objective, except for the release date component) and scheduling more of the first one over the second. If the second job j' has *even slightly smaller* processing time than j has, we would run out of space trying to schedule an equal fraction of j over j', and this loss may hurt us in the (hard) profit requirement. In such a case, we could try to schedule j' over j, observing that the later job j' would not advance too much in time, since j and j' were consecutive in that class and have similar processing times—the eventual hope being that given a small violation of the release dates, we may be able to shift the entire schedule by a bit and regain feasibility.

But this strategy could lead to arbitrarily bad approximations: we could keep fractionally growing a job j until (say) 2/3 of it is scheduled, only to meet a job j' subsequently that also has 2/3 of it scheduled, but j' has smaller processing time and therefore needs to be scheduled over j. In this case, j would shrink to 1/3, and then would start growing again—and repeated occurrences of this might cause the flow time for j to be very high. Indeed, trying to avoid such situations leads us to our algorithm, where we look at a *window* of jobs and select an appropriate one to schedule, rather than greedily running a swapping process. To analyze our algorithm, we charge the total increase in the fractional flow time to the *fractional makespan* of the LP solution, and show that each class of jobs charges the fractional makespan at most twice.

4.3 Notation and Preliminaries

We partition the collection of jobs into *classes*, with jobs in class $\mathcal{C}_k$ having $p_j \in (2^{k-1}, 2^k]$. Notice that $\widetilde{p}_j = 2^k$ for every $j \in \mathcal{C}_k$, and the class of interest with highest index is $\mathcal{C}_{k^*}$. Given a fractional solution

(x, y, f), we say that job j is *fully scheduled* if $y_j = 1$, and *dropped* if $y_j = 0$; in both cases, j is *integrally scheduled*. Let $\mathsf{flow}(x, y, f) = \sum_{j=1}^{n} f_j$ be the fractional cost; note that this is *not the same* as the actual flow time given by this solution, but rather an approximation. Let $\mathcal{P}(x, y, f) = \sum_{j=1}^{n} \sum_{t=0}^{T} x_{jt}$ be the total fractional processing time. Since each job j gets x_{jt} amount of processing time in $[t, t+1)$, the cost of (x, y, f) remains unchanged if all jobs are processed during the first part $[t, t + \sum_{j=1}^{n} x_{jt})$ of this unit interval; we therefore refer to $[t + \sum_{j=1}^{n} x_{jt}, t+1)$ as the *free time interval* in $[t, t+1)$.

We say that an LP solution (x, y, f) is "non-alternating" across each class if the fractional schedule does not alternate between two jobs of the same class. Formally, the schedule is "non-alternating" if for class k and any two class-k jobs j and j', if $y_j, y_{j'} > 0$ and $r_j < r_{j'}$ (or $r_j = r_{j'}$ and $j < j'$), then for any times t, t' such that $x_{jt} > 0$ and $x_{j't'} > 0$, it holds that $t \leq t'$. We call a solution "packed" if there is no free time between the release date of a job, and the last time it is scheduled by the LP solution. The following lemma is proved in Appendix A.1; we assume that we start off with such a solution.

Lemma 4.3. *There is an optimal LP solution (x^*, y^*, f^*) that is non-alternating and packed.*

4.4 The Flow-Time Rounding Algorithm

At a high level, the rounding algorithm proceeds in two stages.

- In Stage I, for each k, we completely schedule almost as many class-k jobs as the LP does fractionally (up to an additive two jobs). The main challenge, as sketched above, is to do this with only a small change in the fractional flow time and the processing time of these jobs.
- In Stage II, we add in at most two class-k jobs to compensate for the loss of jobs in Stage I. Since we add only two jobs per class, we can show that the additional flow time can be controlled.

4.4.1 Flow-Time Rounding: Stage I

Recall that we want to convert the non-alternating and packed optimal solution (x^*, y^*, f^*) returned by the LP into a new solution (x', y', f') where at least $\lfloor \sum_{j \in \mathcal{C}_k} y_j^* \rfloor - 1$ class-k jobs are *completely scheduled*. The algorithm operates on the classes one by one. For each class, it performs a *swapping phase* where mass is shifted between jobs in this class (potentially violating release dates), and then does a *shifting phase* to handle all the release-date violations.

Swapping Phase for Class-k. Given the non-alternating and packed solution (x^*, y^*, f^*), we run the algorithm for the swapping phase given in Algorithm 2.

Shifting Phase for Class-k. After the above swapping phase for class-k jobs, we perform a *shifting phase* to handle any violated release dates. Specifically, consider the collection of time intervals occupied either by class-k jobs or by free time—by the process given above, this remains fixed over the execution of the swapping phase. We now shift all class-k jobs to the right by $2 \cdot 2^k$ within these intervals. Of course, we need to prove that this takes care of all release date violations.

4.4.2 Analysis for Stage I

Lemma 4.4. *The following properties hold true at the end of Stage I:*

(i) $\mathcal{P}(x', y', f') \leq 2\mathcal{P}(x^*, y^*, f^*)$
(ii) The fractional flow time satisfies $\mathsf{flow}(x', y', f') \leq 4 \cdot \mathsf{flow}(x^*, y^*, f^*) + 6k^* \mathcal{P}(x^*, y^*, f^*)$.
(iii) The sum of flow times over all fully scheduled jobs is at most $2 \cdot \mathsf{flow}(x', y', F') + k^* \mathcal{P}(x', y', F')$.

The analysis proceeds by a somewhat delicate charging argument and the basic idea is the following. In Step 4 of the algorithm, suppose Δ fraction of a job j_1 is being scheduled over Δ fraction of a job j_k: we will *charge* every point in the interval (r_{j_1}, r_{j_k}) by an amount Δ. In the case when a Δ fraction

Algorithm 2 Class-k Swapping

1: **set** $(x', y', f') := (x^*, y^*, f^*)$. Repeat the steps 2-5 until $\lfloor \sum_{j \in \mathcal{C}_k} y^*_j \rfloor - 1$ class-k jobs are completely scheduled in (x', y', f').

2: **advance** all class-k jobs as much as possible without violating release dates (for jobs already violating release dates, don't advance their starting time any further) within the time intervals that are either free or are occupied by class-k jobs.

3: **let** j_1 the first fractionally scheduled job in the current LP solution (x', y', f'). Let j_{q+1} be the first class-k job scheduled after j_1 which has processing time $p_{j_{q+1}} < p_{j_1}$, and say the class-k jobs that are scheduled between j_1 and j_{q+1} are $j_2, j_3, \ldots, j_q$. Note that all these jobs must have greater processing time than p_{j_1}. Also, let free denote the total free time between j_1 and j_{q+1} in the current schedule.

4: **if** $\sum_{k=2}^{q} y'_k + \mathsf{free}/p_{j_1} \geq 1 - y'_{j_1}$, then we know that j_1 can be *completely* scheduled over the jobs $j_2, j_3, \ldots, j_q$ and the free time; **for** $k = 2$ to q, do the following

4a: **if** there is some free time (of total length, say, L) between j_{k-1} and j_k, schedule a fraction $\Delta = \min(1 - y'_{j_1}, L/p_{j_1})$ of j_1 in the free time, and delete a fraction Δ from class-k jobs at the rear end of the schedule. Update (x', y', f').

4b: **schedule** a fraction $\Delta = \min(1 - y'_{j_1}, y'_{j_k})$ of j_1 over a fraction Δ of job j_k (possibly creating some free space). Update (x', y', f').

4c: **if** $k = q$ and there is some free time (of total length, say, L) between j_q and j_{q+1}, schedule a fraction $\Delta = \min(1 - y'_{j_1}, L/p_{j_1})$ of j_1 in the free time, and delete a fraction Δ from class-k jobs at the rear end of the schedule. Update (x', y', f').

5: **else if** $\sum_{k=2}^{q} y'_k + \mathsf{free}/p_{j_1} < 1 - y'_{j_1}$, do the following

- delete a total fraction $\min(\sum_{k=1}^{q} y'_k, y'_{j_{q+1}})$ from a prefix of jobs $j_1, j_2, \ldots, j_q$, and *advance* the current fractional schedule of the job j_{q+1} to occupy the space created. Update the solution (x', y', f'). Note that it may or may not have been possible to schedule j_1 in the space fractionally occupied by jobs $j_2, j_3, \ldots, j_q$ and free time in this interval; for accounting reasons we do the same thing in both cases.

of j_1 is being scheduled over an interval of free time beginning at t, we will then charge every point in the interval (r_{j_1}, t) by the fraction Δ. We then go on to show that $\mathsf{flow}(x', y', f') - \mathsf{flow}(x^*, y^*, f^*)$ is not too much more than the total charge accumulated by the interval $[0, T]$ (recall that T is the last time at which the LP scheduled some fractional job). To complete the proof, we argue that the total charge accumulated is $O(\log P)\mathcal{P}(x^*, y^*, f^*)$.

In Appendix A.2 we restate Stage I in a slightly different way, where we also define the charging process charge associated with each step of the algorithm, and give the complete proof of Lemma 4.4.

4.4.3 Flow-Time Rounding: Stage II

The fractional solution (x', y', F') may not be feasible, since we have only scheduled $\lfloor \sum_{j \in \mathcal{C}_k} y^*_j \rfloor - 1$ jobs from class-k. Hence, for each class-k, arbitrarily pick the minimum number of non-fully-scheduled jobs to bring this number to $\lceil \sum_{j \in \mathcal{C}_k} y^*_j \rceil$ (at most two per class). These jobs are preemptively scheduled as soon as possible after their release date. Since at most two jobs per class are added, the flow time does not change much.

Lemma 4.5. *The sum of the flow times of all added jobs is at most* $k^*(\mathcal{P}(x', y', F') + 2^{k^*+2})$.

Proof. For a class-k, we may have to complete two additional jobs. When we schedule an extra job as soon as possible, it waits only for jobs that were fully scheduled during stage II or for jobs that were added in previous iterations of the current stage. Therefore, its flow time can be at most $\mathcal{P}(x', y', F') + 2\sum_{k=1}^{k^*} 2^k$, and therefore the total flow time of added jobs is at most $k^*(\mathcal{P}(x', y', F') + 2^{k^*+2})$. ∎

Because f_j is lowerbounded by $\sum_t x^*_{jt}/2$, we have that $\mathcal{P}(x^*, y^*, f^*) \leq 2 \cdot \mathsf{Opt}$. Therefore, Lemmas 4.4 and 4.5 in conjunction with the inequalities $\mathcal{P}(x', y', F') \leq 2\mathcal{P}(x^*, y^*, f^*) \leq 4 \cdot \mathsf{Opt}$ and $k^* \leq \log P + 1$, prove the following result for minimizing flow time on a single machine.

Theorem 4.6. *The problem of minimizing flow time on a single machine with unit profits can be approximated within a factor of $O(\log P)$.*

4.5 Flow-Time: Identical Parallel Machines

We conclude this section by showing how to combine our single machine algorithm along with ideas drawn from [13] to obtain an $O(\log P)$ approximation for the case of identical machines. We begin by solving a natural extension of the single machine LP to the setting of identical machines; let (x^*, y^*, f^*) be the resulting fractional solution.

- **Stage I:** We rearrange the jobs to make the schedule non-migratory, while preserving the fraction to which each job has been scheduled. This modification is done by performing the procedure given in [13] with the only change being that the jobs in (x^*, y^*, f^*) are fractionally scheduled. The resulting solution $(\hat{x}, \hat{y}, \hat{f})$ can be shown to have an LP cost of at most $\mathsf{flow}(x^*, y^*, f^*) + O(\log P)\mathcal{P}(x^*, y^*, f^*)$.
- **Stage II:** For each machine, we execute Stage I of the single machine algorithm. As a result, almost all jobs are now integrally scheduled, leaving at most two fractionally scheduled jobs per class and machine, without increasing the LP cost by much. If (x', y', f') denotes the LP solution after this stage, we can show that $\mathsf{flow}(x', y', f') \leq 2 \cdot \mathsf{flow}(x^*, y^*, f^*) + 6 \log P \cdot \mathcal{P}(\hat{x}, \hat{y}, \hat{f})$ (The analysis is identical to that for Stage I of the single machine algorithm, presented in Appendix 4.4.1).
- **Stage III:** We consider all fractionally scheduled class-k jobs (there are at most 2 per machine) and schedule more of the job with least processing time, while deleting an equal fraction from the one with largest processing time, until either the small job is fully scheduled or the large job has been completely dropped. This procedure is repeated till only at most one fractional class-k job remains. The entire process is repeated for each class. In Appendix A.3, we show that the LP cost satisfies $\mathsf{flow}(\widetilde{x}, \widetilde{y}, \widetilde{f}) \leq \mathsf{flow}(x', y', f') + 2 \log P \cdot \mathcal{P}(x', y', f')$ where $(\widetilde{x}, \widetilde{y}, \widetilde{f})$ denotes the solution after Stage III. Subsequently, bounding the actual sum of flow times (of the integrally scheduled jobs) also follows closely to Lemma A.6.
- **Stage IV:** As in the single machine case, we schedule the one remaining fractional job for each class by adding processing time whenever possible. This change does not significantly increase the solution cost much like Lemma 4.5.

The algorithm and the proofs depend heavily on the single machine case; we give more details in Appendix A.3. To conclude, we have the following theorem.

Theorem 4.7. *The problem of minimizing flow time on identical machines with unit profits can be approximated within a factor of $O(\log P)$.*

Acknowledgements: We would like to thank Nikhil Bansal, Chandra Chekuri, Kirk Pruhs, Mohit Singh, and Gerhard Woeginger for useful discussions. We would also like to thank the anonymous reviewers of an earlier version of this paper for several helpful comments.

References

[1] Lectures on proof verification and approximation algorithms. Lecture Notes in Computer Science 1367. Springer.

[2] K. R. Baker. *Introduction to Sequencing and Scheduling.* John Wiley and Sons, Inc., 1974.

[3] N. Bansal, A. Blum, S. Chawla, and K. Dhamdhere. Scheduling for flow-time with admission control. In *ESA '03*, pages 43–54, 2003.

[4] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, and L. Stougie. Multiprocessor scheduling with rejection. In *SODA '96*, pages 95–103, 1996.

[5] R. D. Carr, L. Fleischer, V. J. Leung, and C. A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *SODA '00*, pages 106–115, 2000.

[6] M. Charikar and S. Khuller. A robust maximum completion time measure for scheduling. In *SODA'06*, pages 324–333, 2006.

[7] M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan. Algorithms for facility location problems with outliers. In *SODA '01*, pages 642–651, 2001.

[8] K. Chen. A constant factor approximation algorithm for k-median clustering with outliers. In *SODA'08*, pages 826–835, 2008.

[9] F. A. Chudak, T. Roughgarden, and D. P. Williamson. Approximate k-msts and k-steiner trees via the primal-dual method and lagrangean relaxation. *Mathematical Programming*, 100(2):411–421, 2004.

[10] D. W. Engels, D. R. Karger, S. G. Kolliopoulos, S. Sengupta, R. N. Uma, and J. Wein. Techniques for scheduling with rejection. In *ESA'98*, pages 175–191, 2003.

[11] L. Epstein, J. Noga, and G. J. Woeginger. On-line scheduling of unit time jobs with rejection: minimizing the total completion time. *Operations Research Letters*, 30(6):415–420, 2002.

[12] N. Garg. Saving an epsilon: a 2-approximation for the k-MST problem in graphs. In *STOC'05*, pages 396–402, 2005.

[13] N. Garg and A. Kumar. Better algorithms for minimizing average flow-time on related machines. In *ICALP '06*, pages 181–190, 2006.

[14] N. Garg and A. Kumar. Minimizing average flow-time : Upper and lower bounds. In *FOCS '07*, pages 603–613, 2007.

[15] D. Golovin, V. Nagarajan, and M. Singh. Approximating the k-multicut problem. In *SODA '06*, pages 621–630, 2006.

[16] S. Guha and K. Munagala. Model-driven optimization using adaptive probes. In *SODA '07*, pages 308–317, 2007.

[17] L. A. Hall, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: off-line and on-line algorithms. In *SODA '96*, pages 142–151, 1996.

[18] H. Hoogeveen, M. Skutella, and G. J. Woeginger. Preemptive scheduling with rejection. *Mathematical Programming, Ser. B*, 94(2-3):361–374, 2003.

[19] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

[20] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. In M. J. Atallah, editor, *Handbook of Algorithms and Theory of Computation*. CRC Press, 1997.

[21] J. Könemann, O. Parekh, and D. Segev. A unified approach to approximating partial covering problems. In *ESA'06*, pages 468–479, 2006.

[22] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[23] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *STOC '97*, pages 110–119, 1997.

[24] A. Levin and D. Segev. Partial multicuts in trees. *Theoretical Computer Science*, 369(1-3):384–395, 2006.

[25] J. Mestre. A primal-dual approximation algorithm for partial vertex cover: making educated guesses. In *APPROX '05*, pages 182–191. 2005.

[26] J. Mestre. Lagrangian relaxation and partial cover (extended abstract). In *STACS '08*, pages 539–550, 2008.

[27] A. S. Schulz and M. Skutella. Scheduling unrelated machines by randomized rounding. *SIAM Journal on Discrete Mathematics*, 15(4):450–469, 2002.

[28] S. S. Seiden. Preemptive multiprocessor scheduling with rejection. *Theoretical Computer Science*, 262(1):437–458, 2001.

[29] D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993.

[30] M. Skutella. Convex quadratic and semidefinite programming relaxations in scheduling. *Journal of the ACM*, 48(2):206–242, 2001.

[31] L. A. Wolsey. Faces for a linear inequality in 0-1 variables. *Mathematical Programming*, 8:165–178, 1975.

A Proofs from Section 4

A.1 Proof of Lemma 4.3: Non-Alternating and Compact Optimal Solutions

Consider doing the following changes for every class $1 \leq k \leq k^*$ in some arbitrary order.

1. Let $\mathcal{T}(k)$ be the union of all time intervals in which the LP solution (x^*, y^*, f^*) schedules class-k jobs along with the overall free time.
2. Use $\mathcal{T}(k)$ to continuously schedule a y^*_j fraction of each job $j \in \mathcal{C}_k$. These fractions are scheduled in increasing order of release dates as soon as possible (while respecting release dates).

Let $(\widehat{x}, \widehat{y}, \widehat{f})$ be the resulting LP solution after we finish this operation. Notice that this solution is non-alternating and packed. Also, every job is still scheduled to the same extent as before, i.e. $\widehat{y}_j = y^*_j$, and consequently, we have, $\mathcal{P}(\widehat{x}, \widehat{y}, \widehat{f}) = \mathcal{P}(x^*, y^*, F^*)$. It remains to show that $\mathsf{flow}(\widehat{x}, \widehat{y}, \widehat{f}) \leq \mathsf{flow}(x^*, y^*, f^*)$. Let $V_{k,t}(x, y, f) = \sum_{j \in \mathcal{C}_k} \sum_{t'=t}^{T} x_{jt'}$ be the overall processing time of class-k jobs after time t. Clearly since we are only rearranging class-k jobs (and subsequently advancing jobs whenever possible) within the time intervals in which they were scheduled in (x^*, y^*, f^*), we have $V_{k,t}(\widehat{x}, \widehat{y}, \widehat{f}) \leq V_{k,t}(x^*, y^*, f^*)$ for every k and t. Now,

$$\begin{aligned}
\mathsf{flow}(\widehat{x}, \widehat{y}, \widehat{f}) - \mathsf{flow}(x^*, y^*, f^*) &= \sum_{j=1}^{n} \left(\widehat{f}_j - f^*_j \right) \\
&= \sum_{k=1}^{k^*} \sum_{j \in \mathcal{C}_k} \sum_{t=0}^{T} \left(\left(\frac{\widehat{x}_{jt}}{\widetilde{p}_j} \left(t + \frac{1}{2} - r_j \right) + \frac{\widehat{x}_{jt}}{2} \right) - \left(\frac{x^*_{jt}}{\widetilde{p}_j} \left(t + \frac{1}{2} - r_j \right) + \frac{x^*_{jt}}{2} \right) \right) \\
&= \sum_{k=1}^{k^*} \sum_{j \in \mathcal{C}_k} \sum_{t=0}^{T} \frac{t(\widehat{x}_{jt} - x^*_{jt})}{\widetilde{p}_j} \\
&= \sum_{k=1}^{k^*} \frac{1}{2^k} \sum_{t=0}^{T} t \sum_{j \in \mathcal{C}_k} (\widehat{x}_{jt} - x^*_{jt}) \\
&= \sum_{k=1}^{k^*} \frac{1}{2^k} \sum_{t=0}^{T} \left(V_{k,t}(\widehat{x}, \widehat{y}, \widehat{f}) - V_{k,t}(x^*, y^*, f^*) \right) \leq 0
\end{aligned}$$

The second equality holds since $\sum_{t=0}^{T} \widehat{x}_{jt} = \sum_{t=0}^{T} x^*_{jt}$ for every j. The last equality holds since $\sum_{t=0}^{T} t \sum_{j \in \mathcal{C}_k} \widehat{x}_{jt} = \sum_{t=0}^{T} V_{k,t}(\widehat{x}, \widehat{y}, \widehat{f})$ and $\sum_{t=0}^{T} t \sum_{j \in \mathcal{C}_k} x^*_{jt} = \sum_{t=0}^{T} V_{k,t}(x^*, y^*, f^*)$.

A.2 Proof of Lemma 4.4: The Stage I Algorithm

The first part is simple to prove. Observe that the only step where the Stage I algorithm would schedule a larger job while deleting a smaller job is in Step 4a (or Step 4c). However, even in this case, since it

operates within a particular class, the worst case would be scheduling Δ fraction of a job j in some free time, while deleting Δ fraction of a job j' whose processing time is half that of j. Therefore, it follows that $\mathcal{P}(x', y', f') \leq 2\mathcal{P}(x^*, y^*, f^*)$.

For the proof of the next two parts, we describe an equivalent form of the swapping algorithm which operates on each time slot $[t, t+1)$ one by one, rather than job by job. As the algorithm proceeds, the solution (x', y', f') keeps getting updated: initially $(x', y', f') = (x^*, y^*, f^*)$. We introduce a *charging scheme* $\mathsf{charge}(t, j)$ which is initially set to 0 for every $t \in [0, T]$ and $j \in \{1, \ldots, n\}$. It is progressively modified over the course of the algorithm, and helps us bound the increase in LP cost as the jobs are make integral.

Revisiting the Swapping phase for class-k. If the number of fully scheduled class-k jobs in the current solution is smaller than $\lfloor \sum_{j \in \mathcal{C}_k} y'_j \rfloor - 2$, we first *advance* all class-k jobs as much as possible within the union of class-k time intervals along with the overall free time. That is, we make sure that there is no free time between the release date of any fractionally scheduled class-k job and the last time interval in which some fraction of it is scheduled. Note that this does not cause any increase in cost of the LP solution.

Now consider some stage of the swapping stage where j_1 be the first fractionally scheduled class-k job, and let $j_2, \ldots, j_q$ be a prefix of the class-k jobs scheduled after j_1 defined thus: q is the minimal index for which $p_{j_{q+1}} < p_{j_1}$ or for which j_q is the last scheduled class-k job (see Figure 2). For any $s < q$, let $\mathsf{free}(j_s, j_{s+1})$ be the overall amount of free time between the last interval in which j_s is scheduled and the first interval in which j_{s+1} is scheduled[2]. There are two cases to consider:

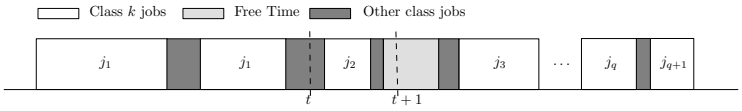

Figure 2: A prefix of class-k jobs (schematic illustration).

Case I: $\sum_{s=2}^{q} y'_{j_s} + \sum_{s=1}^{q} \mathsf{free}(j_s, j_{s+1})/p_{j_1} \geq 1 - y'_{j_1}$. In this scenario, repeat until j_1 becomes fully scheduled:

1. Let $s \in \{2, \ldots, q\}$ be the minimal index for which $y'_{j_s} > 0$. If $y'_{j_2} = \cdots = y'_{j_q} = 0$, let $s = q + 1$.
2. If $\mathsf{free}(j_1, j_s) = 0$, let $[t, t+1)$ be the first time slot where j_s is scheduled. We now replace a $\Delta_t = \min\{1 - y'_{j_1}, x'_{j_s t}/p_{j_s}\}$ fraction of j_s by a Δ_t fraction of j_1, possibly creating some free time. We also advance all class-k jobs starting from j_s as much as possible (without violating release dates) within the union of all time intervals in which these jobs are scheduled along with the overall free time. An illustration of this step (before the advancement) is shown in Figure 3.

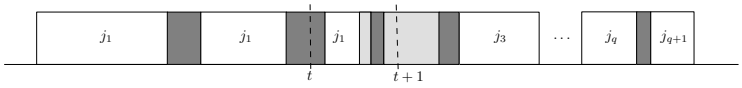

Figure 3: Example of Case I. Notice the free time created due to p_{j_2} being smaller than p_{j_1}.

Charging: When a Δ_t fraction of j_s is replaced by a Δ_t fraction of j_1, we say that *each point* of the time interval (r_{j_1}, r_{j_s}) pays $\Delta_t p_{j_1}/\widetilde{p}_{j_1}$ *on behalf* of j_1. I.e., set $\mathsf{charge}(t', j_1) \leftarrow \mathsf{charge}(t', j_1) + \Delta_t p_{j_1}/\widetilde{p}_{j_1}$ for all $t' \in (r_{j_1}, r_{j_s})$.

[2] If j_q is the last scheduled class-k job, we define $\mathsf{free}(j_q, j_{q+1}) = \infty$.

Cost increment: When we replace j_s with j_1, the extra LP cost paid by j_1 is $\frac{\Delta_t p_{j_1}}{\widetilde{p}_{j_1}}\left(t+\frac{1}{2}-r_{j_1}\right)+\frac{\Delta_t p_{j_1}}{2}$ whereas the cost saved by removing a fraction of j_s is $\frac{\Delta_t p_{j_s}}{\widetilde{p}_{j_s}}\left(t+\frac{1}{2}-r_{j_s}\right)+\frac{\Delta_t p_{j_s}}{2}$. The cost increase is

$$\left(\frac{\Delta_t}{\widetilde{p}_{j_1}}\left(t+\tfrac{1}{2}-r_{j_s}\right)+\tfrac{\Delta_t}{2}\right)(p_{j_1}-p_{j_s})+\frac{\Delta_t p_{j_1}}{\widetilde{p}_{j_1}}(r_{j_s}-r_{j_1})\le\frac{\Delta_t p_{j_1}}{\widetilde{p}_{j_1}}(r_{j_s}-r_{j_1})\,,$$

which is exactly the increment in $\int_0^T\sum_{j\in\mathcal{C}_k}\mathsf{charge}(t,j)dt$. The inequality above holds because $p_{j_s}\ge p_{j_1}$.

3. Otherwise (i.e., $\mathsf{free}(j_1,j_s)>0$) let I be the first free time interval between the last interval in which j_1 is scheduled and the first interval in which j_s is scheduled. Also, let $[t,t+1)$ be the first time slot having a non-empty intersection with I, and let j_l be the last fractionally scheduled class-k job. Note that j_l cannot be any of the jobs $j_2,\ldots,j_{q+1}$, as the number of fully scheduled class-k jobs is at most $\lfloor\sum_{j\in\mathcal{C}_k}y'_j\rfloor-2$. Now, we schedule an extra $\Delta_t=\min\{1-y'_{j_1},y'_{j_l},|I\cap[t,t+1)|/p_{j_1}\}$ fraction of j_1 in $I\cap[t,t+1)$, while continuously deleting a Δ_t fraction of j_l from the intervals where this job is scheduled, in reverse order of time.

Charging: If $r_{j_1}\le t$, each point of the time interval (r_{j_1},t) pays $\Delta_t p_{j_1}/\widetilde{p}_{j_1}$ on behalf of j_1. That is, we set $\mathsf{charge}(t',j_1)\leftarrow\mathsf{charge}(t',j_1)+\Delta_t p_{j_1}/\widetilde{p}_{j_1}$ for every $t'\in(r_{j_1},t)$.

Cost increment: The extra cost paid by j_1 is $\frac{\Delta_t p_{j_1}}{\widetilde{p}_{j_1}}\left(t+\frac{1}{2}-r_{j_1}\right)+\frac{\Delta_t p_{j_1}}{2}$, while the cost saved by deleting a Δ_t fraction of j_l is at least $\Delta_t p_{j_l}/2$. Since j_1 and j_l belong to the same class, the cost increment is at most

$$\frac{\Delta_t p_{j_1}}{\widetilde{p}_{j_1}}(t-r_{j_1})+3\cdot\frac{\Delta_t p_{j_l}}{2}\ .$$

When $r_{j_1}\le t$, the first term is exactly the increment in $\int_0^T\sum_{j\in\mathcal{C}_k}\mathsf{charge}(t,j)dt$ as a result of setting $\mathsf{charge}(t',j_1)\leftarrow\mathsf{charge}(t',j_1)+\Delta_t p_{j_1}/\widetilde{p}_{j_1}$ for every $t'\in(r_{j_1},t)$. In the opposite case, this term is negative, which is why we do not need to modify the charging function. In addition, the term $\Delta_t p_{j_l}/2$ lower bounds the contribution of the deleted fraction of j_l towards the quantity $\mathsf{flow}(x^*,y^*,f^*)$.

Case II: $\sum_{s=2}^q y'_{j_s}+\sum_{s=1}^q\mathsf{free}(j_s,j_{s+1})/p_{j_1}<1-y'_{j_1}$. In this case, repeat until j_{q+1} becomes fully scheduled or until $y'_{j_1}=\cdots=y'_{j_q}=0$:

1. Let $s\in\{1,\ldots,q\}$ be the minimal index for which $y'_{j_s}>0$.
2. Let $[t,t+1)$ be the first time slot where j_s is scheduled. We now replace a $\Delta_t=\min\{1-y'_{j_{q+1}},x'_{j_s t}/p_{j_s}\}$ fraction of j_s by a Δ_t fraction of j_{q+1}, possibly creating some free time.

Cost increment: The extra cost paid by j_{q+1} is $\frac{\Delta_t p_{j_{q+1}}}{\widetilde{p}_{j_{q+1}}}\left(t+\frac{1}{2}-r_{j_{q+1}}\right)+\frac{\Delta_t p_{j_{q+1}}}{2}$, whereas the cost saved by scheduling a smaller fraction of j_s is $\frac{\Delta_t p_{j_s}}{\widetilde{p}_{j_s}}\left(t+\frac{1}{2}-r_{j_s}\right)+\frac{\Delta_t p_{j_s}}{2}$. Since $p_s\ge p_{j_1}>p_{j_{q+1}}$, the cost increment is

$$\left(\frac{\Delta_t}{\widetilde{p}_{j_s}}\left(t+\tfrac{1}{2}-r_{j_s}\right)+\tfrac{\Delta_t}{2}\right)(p_{j_{q+1}}-p_{j_s})+\frac{\Delta_t p_{j_{q+1}}}{\widetilde{p}_{j_s}}(r_{j_s}-r_{j_{q+1}})\le 0\ ,$$

and there is no need to modify the charging function.

Conclude this case by making the following rearrangements:

1. Continuously schedule a $y'_{j_{q+1}}$ fraction of j_{q+1} within the union of class-k intervals and free time, starting at the first interval in which j_{q+1} is currently processed. Even though we may have violated the release date of j_{q+1}, the earliest time in which any part of this job is processed was advanced by at most $2\cdot 2^k$ within the union of time intervals where class-k jobs are scheduled and free time intervals, since we initially had $\sum_{s=2}^q y'_{j_s}+\sum_{s=1}^q\mathsf{free}(j_s,j_{s+1})/p_{j_1}<1-y'_{j_1}$. This anomaly will be handled in the sequel.

2. Following j_{q+1}, proceed by scheduling a $y'_{j_1}, \ldots, y'_{j_q}$ fraction of $j_1, \ldots, j_q$, respectively, as soon as possible (without violating release dates) within the time intervals where class-k jobs are scheduled and free time.

We are now ready to prove Lemma 4.4. Noting that the fractional contribution of each class changes only during the course of its corresponding iteration, we may focus our attention on a fixed class-k, and bound its fractional cost, $\mathsf{flow}_k(x', y', f') = \sum_{j \in \mathcal{C}_k} F'_j$.

Claim A.1. *Just before the shifting phase for class-k, we have*

$$\mathsf{flow}_k(x', y', f') \leq 4 \cdot \mathsf{flow}_k(x^*, y^*, f^*) + \int_0^T \sum_{j \in \mathcal{C}_k} \mathsf{charge}(t, j) dt \ .$$

Proof. We can bound the cost increment of each operation in the swapping phase as follows:

- A single operation in case I, step 2: As mentioned in the algorithm, the cost increment is upper bounded by the increment in $\int_0^T \sum_{j \in \mathcal{C}_k} \mathsf{charge}(t, j) dt$.
- A single operation in case I, step 3: In these settings, the cost increment can be bounded by the increment in $\int_0^T \sum_{j \in \mathcal{C}_k} \mathsf{charge}(t, j) dt$ plus thrice whatever the deleted fraction of j_l contributes to $\mathsf{flow}(x^*, y^*, f^*)$. It is important to observe that deleted fractions will not be used later on to bound additional cost increments for subsequent operations for this stage.
- Operations in case II: Because we are only rearranging jobs within class-k space (and never introduce any free time), arguments similar to the proof in Appendix A.1 show that no extra cost is incurred in this step.

Therefore, at the completion of the swapping phase, we have

$$\mathsf{flow}_k(x', y', f') - \mathsf{flow}_k(x^*, y^*, f^*) \leq \int_0^T \sum_{j \in \mathcal{C}_k} \mathsf{charge}(t, j) dt + 3 \cdot \mathsf{flow}_k(x^*, y^*, f^*) \ .$$

■

We proceed by establishing a few crucial properties of the charging function.

Claim A.2. *Just before the shifting phase, no free time ever pays on behalf of any job. In other words, if $\sum_{j \in \mathcal{C}_k} \mathsf{charge}(t, j) > 0$ then t cannot be free time.*

Proof. We prove the above claim by arguing that, whenever an interval is charged, each of its points is currently dedicated to processing some job. It is not difficult to verify that our algorithm preserves this property till the swapping phase terminates. Consider an operation where some interval is charged.

- In case I, step 2, suppose that a Δ_t fraction of j_s is replaced by a Δ_t fraction of j_1. Then, the charging scheme increases $\mathsf{charge}(t', j_1)$ by $\Delta_t p_{j_1} / \widetilde{p}_{j_1}$ for every $t' \in (r_{j_1}, r_{j_s})$. However, we are guaranteed not to have free time in the interval (r_{j_1}, r_{j_s}) because of the fact that there is no free time between r_{j_1} and the last interval in which j_1 is scheduled (any such free time is eliminated when we begin the swapping phase for class-k), and because $\mathsf{free}(j_1, j_s) = 0$.
- In case I, step 3, the algorithm picks the first time slot (say, $[t_1, t_1 + 1)$) which has some free time between the last interval in which j_1 is scheduled and the first interval in which j_s is scheduled. Suppose that an extra Δ_{t_1} fraction of j_1 is scheduled, increasing $\mathsf{charge}(t', j_1)$ by $\Delta_{t_1} p_{j_1} / \widetilde{p}_{j_1}$ for every $t' \in (r_{j_1}, t_1)$. Note that there cannot be free time between the last interval in which j_1 is scheduled and t_1 (by the way $[t_1, t_1 + 1)$ was picked), and also between r_{j_1} and the last interval in which j_1 is scheduled. Therefore, there is no free time in (r_{j_1}, t_1).

■

Claim A.3. *Each point in time pays on behalf of at most one job. That is, for every $t \in [0, T]$,*

$$|\{j \in \mathcal{C}_k : \mathsf{charge}(t, j) > 0\}| \leq 1 \ .$$

Proof. We prove the above claim by contradiction. For this purpose, suppose there exists a point in time $t^* \in [0, T]$ that pays on behalf of two jobs, say j and j'. By the way we charged jobs in the swapping phase, we know that both j and j' must be fully scheduled. Without loss of generality, we assume that j appears before j' in the schedule (x^*, y^*, f^*).[3] Consider a single operation in which t^* is charged, paying some amount of behalf of j.

- Suppose step 2 is executed in time slot $[t, t+1)$, where Δ_t fraction of a job j_s is replaced by Δ_t fraction of j during which t^* pays on behalf of j. Since t^* is charged in this operation, it follows that $t^* \in (r_j, r_{j_s})$. In addition, we have $r_{j_s} \leq r_{j'}$, or otherwise j' must have been fully replaced by j during previous operations, since j is currently replacing j_s. Therefore, $t^* < r_{j_s} \leq r_{j'}$, implying that t^* cannot be paying on behalf of j', since our charging scheme guarantees that a time point can pay on behalf of a particular job only when it appears after the release date of this job.
- On the other hand, suppose step 3 is executed in time slot $[t, t+1)$, where a Δ_t fraction of j is scheduled. Since t^* is charged, it follows that $t^* \in (r_j, t)$. Now, if t^* pays on behalf of j', we must have $r_{j'} < t^* < t$, meaning that at the moment there is some free time between $r_{j'}$ and the first interval in which j' is scheduled. Such free time would have been eliminated at the beginning of this iteration as a result of advancing class-k jobs.

■

Claim A.4. $\int_0^T \sum_{j \in \mathcal{C}_k} \mathsf{charge}(t, j) dt \leq 2\mathcal{P}(x^*, y^*, f^*)$.

Proof. By Claim A.3, we know that each point in time pays on behalf of at most one job per class. Also, whenever a point t is charged on behalf of a job j, the increment in $\mathsf{charge}(t, j)$ is of the form $\Delta p_j / \widetilde{p}_j$, where Δ is the additional fraction of j being scheduled. Therefore, the total amount t can pay on behalf of j is at most $p_j / \widetilde{p}_j \leq 1$. This bound, coupled with the observation that free time never pays on behalf of any job (see Claim A.2), proves that

$$\int_0^T \sum_{j \in \mathcal{C}_k} \mathsf{charge}(t, j) dt \leq \mathcal{P}(x', y', f') \leq 2\mathcal{P}(x^*, y^*, f^*) \ .$$

The last inequality holds since, throughout stage II, the overall processing time cannot grow by a factor greater than 2: Whenever we schedule an extra fraction of some class-k job, we also delete an equal fraction from some other class-k job, which in the worst case has half its processing time. ■

Claim A.5. *Immediately after the shifting phase, we have*

$$\mathsf{flow}_k(x', y', f') \leq 4 \cdot \mathsf{flow}_k(x^*, y^*, f^*) + 6\mathcal{P}(x^*, y^*, f^*) \ .$$

Proof. By combining Claims A.1 and A.4, we can bound the fractional cost of class-k just after the swapping phase by

$$\begin{aligned} \mathsf{flow}_k(x', y', f') &\leq 4 \cdot \mathsf{flow}_k(x^*, y^*, f^*) + \int_0^T \sum_{j \in \mathcal{C}_k} \mathsf{charge}(t, j) dt \\ &\leq 4 \cdot \mathsf{flow}_k(x^*, y^*, f^*) + 2\mathcal{P}(x^*, y^*, f^*) \ . \end{aligned}$$

In addition, arguments nearly identical to those of Garg and Kumar [14, Clm. 4.3] show that the cost increment due to the shifting phase is at most $2\mathcal{P}(x', y', f') \leq 4\mathcal{P}(x^*, y^*, f^*)$. ■

[3] This assumption implies $r_j \leq r_{j'}$ because the LP solution is assumed to be non-alternating.

Part (ii) of Lemma 4.4 is now derived by summing the inequality stated in Claim A.5 over all classes:

$$\begin{aligned}\mathsf{flow}(x', y', f') &= \sum_{k=1}^{k^*} \mathsf{flow}_k(x', y', f') \\ &\leq 4 \sum_{k=1}^{k^*} \mathsf{flow}_k(x^*, y^*, f^*) + 6k^* \mathcal{P}(x^*, y^*, f^*) \\ &= 4 \cdot \mathsf{flow}(x^*, y^*, f^*) + 6k^* \mathcal{P}(x^*, y^*, f^*) \ .\end{aligned}$$

The third part of Lemma 4.4 follows from the next lemma.

Lemma A.6. *The sum of flow times of all integrally scheduled jobs is at most* $2 \cdot \mathsf{flow}(x', y', f') + k^* \mathcal{P}(x', y', f')$.

Proof. Consider some fully scheduled job j. It is easy to verify (see [13] for a proof) that the quantity $2F'_j$ is at least the actual flow time of j minus the amount of time for which j has been preempted (which cannot include free time). In addition, our algorithm ensures that, at any point in time, at most one class-k job may be preempted. Hence, by summing over all fully scheduled class k jobs, it follows that their sum of flow times is bounded by $2 \sum_{j \in \mathcal{C}_k} F'_j + \mathcal{P}(x', y', f') = 2 \cdot \mathsf{flow}_k(x', y', f') + \mathcal{P}(x', y', f')$. The desired result is obtained by summing over all classes. ■

A.3 Parallel Machines Algorithm

For completeness, we first provide the natural extension of the flow time LP for identical machines. Then, we present the algorithm in more detail.

$$\begin{array}{lll}
\text{minimize} & \sum_{j=1}^{n} f_j & \\
\text{subject to} & (1) \quad f_j = \displaystyle\sum_{t=0}^{T} \sum_{i=1}^{m} \left(\frac{x_{ijt}}{\widetilde{p}_j} \left(t + \frac{1}{2} - r_j \right) + \frac{x_{ijt}}{2} \right) & \forall\, j \\
& (2) \quad p_j y_j = \sum_{t=0}^{T} \sum_{i=1}^{m} x_{ijt} & \forall\, j \\
& (3) \quad \sum_{j=1}^{n} x_{ijt} \leq 1 & \forall\, t, i \\
& (4) \quad \sum_{j=1}^{n} y_j \geq \Pi & \\
& (5) \quad x_{ijt} = 0 & \forall\, i, j, t : t < r_j \\
& (6) \quad x_{ijt} \geq 0,\ 0 \leq y_j \leq 1 & \forall\, i, j, t
\end{array}$$

The Algorithm:

Stage I: Let (x^*, y^*, f^*) be an optimal LP solution. We first rearrange the jobs following the procedure given in [13] to make each job's schedule non-migratory whilst preserving the fraction to which it has been scheduled. Let $(\widehat{x}, \widehat{y}, \widehat{f})$ be the updated solution. A proof identical to Lemma 3.3 in [13] shows that $\mathsf{flow}(\widehat{x}, \widehat{y}, \widehat{f}) \leq \mathsf{flow}(x^*, y^*, f^*) + O(\log P)\mathcal{P}(x^*, y^*, f^*)$ and that $\mathcal{P}(\widehat{x}, \widehat{y}, \widehat{f}) \leq \mathcal{P}(x^*, y^*, f^*)$.

Stage II: Let $\mathsf{flow}^i(x, y, f)$ of an LP solution (x, y, f) be $\sum_j \sum_{t=0}^{T} \left(\frac{x_{ijt}}{\widetilde{p}_j} \left(t + \frac{1}{2} - r_j \right) + \frac{x_{ijt}}{2} \right)$, and $\mathcal{P}^i(x, y, f) = \sum_j \sum_{t=0}^{T} x_{ijt}$. For each machine, run Stage I of the single machine algorithm: let (x', y', f') be the (possibly infeasible) solution obtained. From the analysis of the single machine case, we have $\mathsf{flow}^i(x', y', f') \leq 2 \cdot \mathsf{flow}^i(\widehat{x}, \widehat{y}, \widehat{f}) + 6(\log P)\mathcal{P}^i(\widehat{x}, \widehat{y}, \widehat{f})$, and $\mathcal{P}^i(x', y', f') \leq 2\mathcal{P}^i(\widehat{x}, \widehat{y}, \widehat{f})$. Further, for each k, the number of class-k jobs completely scheduled on any machine i in (x', y', f') is at least the fractional number of class-k jobs scheduled on machine i by (x^*, y^*, f^*) (up to an additive 2 jobs).

Stage III: We now handle the infeasibility of (x', y', f'): each class may still have up to 2 fractionally scheduled jobs on each machine. We set $(\widetilde{x}, \widetilde{y}, \widetilde{f}) := (x', y', f')$, and make changes to $(\widetilde{x}, \widetilde{y}, \widetilde{f})$. Like in the single machine case, we *swap* jobs to make them integrally scheduled. For each k,

IIIa: Advance all class-k jobs as much as possible (within time occupied by class-k jobs and free time) such that there is no free time between when a job is released and when it is scheduled in $(\widetilde{x}, \widetilde{y}, \widetilde{f})$.

IIIb: Repeat the following until there is at most 1 fractionally scheduled class-k job in $(\widetilde{x}, \widetilde{y}, \widetilde{f})$:

Let j_1 be the fractionally scheduled class-k job with largest processing time and j_2 be the one with smallest processing time. Keep adding j_2 to the end of the schedule on the machine in which it has currently been scheduled, while deleting an equal fraction from j_1 until either (i) j_2 is fully scheduled, or (ii) j_1 has been completely deleted.

Analysis. Suppose the algorithm is replacing j_1 (scheduled on machine i_1) with j_2 (scheduled on machine i_2). Instead of performing the replacement in one shot, we could also do it in a time slot by time slot basis. Let the last time interval in which j_1 is scheduled be $[t_1, t_1 + 1)$, and let $[t_2, t_2 + 1)$ be first interval that has free time, after the fractional completion of j_2. The algorithm deletes a fraction $\Delta = \min(\widetilde{x}_{i_1 j_1 t}/p_{j_1}, 1 - \widetilde{y}_{j_2}, (1 - \sum_j \widetilde{x}_{i_2 j t_2})/p_{j_2})$ of j_1 and schedules Δ fraction of j_2 in the free time in $[t_2, t_2 + 1)$ on machine i_2. Intuitively, Δ is the minimum of the fraction of j_1 that is scheduled in $[t_1, t_1 + 1)$, the fraction of j_2 needed to make it fully scheduled, and the fraction of j_2 that can be scheduled in the free time in $[t_2, t_2 + 1)$.

Observe that because $p_{j_2} \leq p_{j_1}$, the additional cost incurred by the modified LP solution is at most

$$(t_2 - r_{j_2} + \frac{1}{2})\Delta + \frac{\Delta p_{j_2}}{2} - (t_1 - r_{j_1} + \frac{1}{2})\Delta + \frac{\Delta p_{j_1}}{2} \leq (t_2 - r_{j_2})\Delta_t$$

Also notice that every point in the interval (r_{j_2}, t_2) is not free time, by the way t_2 was chosen.

We then employ a charging scheme where each point t on the time interval (r_{j_2}, t_2) pays an additional charge of Δ towards job j_2. The following properties are then true at the end of this stage:

(a) Each point pays at most 2 on behalf of jobs belonging to a class on each machine. This is because there can be at most 2 fractional jobs per class on each machine in the solution (x', y', f') and a point in time pays only for a fractional job which becomes completely scheduled.
(b) Any point which pays on behalf of a job cannot be "free time". This follows because we are guaranteed that there is no free time in any charging interval.
(c) The total processing time in the LP solution does not increase: $\mathcal{P}(\widetilde{x}, \widetilde{y}, \widetilde{f}) \leq \mathcal{P}(x', y', f')$. This holds because we always replace a fraction of a larger job with an equal fraction of a smaller job.

Therefore, at the end of this stage, the cost of the updated LP solution $(\widetilde{x}, \widetilde{y}, \widetilde{f})$ is bounded by

$$\mathsf{flow}(\widetilde{x}, \widetilde{y}, \widetilde{f}) \leq \mathsf{flow}(x', y', f') + 2(\log P)\mathcal{P}(x', y', f')$$

and the total processing time by

$$\mathcal{P}(\widetilde{x}, \widetilde{y}, \widetilde{f}) \leq \mathcal{P}(x', y', f') \ .$$

Stage IV: After Stage III, we might still have at most one fractional job per class. To handle this, for each class-k, we completely schedule the last remaining fractionally scheduled class-k whenever possible on the machine in which it has been fractionally scheduled. The analysis for this step, and the one for bounding the actual sum of flow times of the integrally scheduled jobs is analogous to the one for the single machine case. This proves Theorem 4.7.

B Average Weighted Completion Time

B.1 Weighted Completion Time with K Profit Constraints

We now consider an extension of the problem studied in Section 3: one in which there are K different profit requirements of the form $\sum_j \pi_j^k y_j \geq \Pi^k$ for $1 \leq k \leq K$. We highlight the changes to be made to our algorithm and then present its analysis.

Necessary modifications:

(a) KC constraints for each profit requirement are written down in the LP. Analogous to the single profit requirement case, we define $\pi_j^{k,\mathcal{A}} = \min\{\pi_j^k, \Pi^k - \Pi^k(\mathcal{A})\}$ for each subset of jobs $\mathcal{A}$ and $1 \leq k \leq K$.

(b) We set $\mathcal{A}^* = \{j : \widehat{y}_j \geq 1/\beta_K\}$. That is, instead of rounding up each $\widehat{y}_j$ by a factor of 2, we round these variables by a factor of β_K, a parameter whose value will be determined later.

(c) For jobs in $\mathcal{A}^*$, we mark each machine/time pair $\tau_j = (i,t)$ with probability $\widehat{x}_{ijt}/(p_{ij}\widehat{y}_j)$. For jobs not in $\mathcal{A}^*$, we mark each machine/time pair $\tau_j = (i,t)$ with probability $\beta_K \widehat{x}_{ijt}/p_{ij}$. Essentially, we pick jobs in $\mathcal{A}^*$ with probability 1, and every other job with probability $\beta_K \widehat{y}_j$.

Analysis. The proof that the expected cost is within a factor of $O(\beta_K)$ of optimal is nearly identical to that of the single profit requirement case. Therefore, we would like to fix β_K such that all profit constraints are simultaneously satisfied with constant‘ probability. To this end, consider one such profit requirement Π^k. We upper bound the probability that the collection of jobs picked does not satisfy this requirement. Consider the knapsack cover inequality for $\mathcal{A}^*$ with respect to requirement k, stating that $\sum_{j \notin \mathcal{A}^*} \pi_j^{k,\mathcal{A}^*} \widehat{y}_j \geq \Pi^k - \Pi^k(\mathcal{A}^*)$. The total profit collected from jobs not in $\mathcal{A}^*$ can be lower bounded by $Z = \sum_{j \notin \mathcal{A}^*} \pi_j^{k,\mathcal{A}^*} Z_j$; here, each Z_j is a random variable indicating whether job j is picked. To provide an upper bound on the probability that Z falls below $\Pi^k - \Pi^k(\mathcal{A}^*)$, note that

$$\mathrm{E}[Z] = \mathrm{E}\left[\sum_{j \notin \mathcal{A}^*} \pi_j^{k,\mathcal{A}^*} Z_j\right] = \beta_K \sum_{j \notin \mathcal{A}^*} \pi_j^{k,\mathcal{A}^*} \widehat{y}_j \geq \beta_K(\Pi^k - \Pi^k(\mathcal{A}^*)) \ .$$

Consequently, let us define $\alpha_j = \pi_j^{k,\mathcal{A}^*}/(\Pi^k - \Pi^k(\mathcal{A}^*))$. Since our algorithm *independently* picks each job not in $\mathcal{A}^*$ with probability $\beta_K \widehat{y}_j$, we have

$$\begin{aligned}\Pr\left[Z \leq \Pi^k - \Pi^k(\mathcal{A}^*)\right] &= \Pr\left[\sum_{j \notin \mathcal{A}^*} \frac{\pi_j^{k,\mathcal{A}^*}}{\Pi^k - \Pi^k(\mathcal{A}^*)} Z_j \leq 1\right] \leq \Pr\left[\sum_{j \notin \mathcal{A}^*} \alpha_j Z_j \leq \frac{\mathrm{E}[\sum_{j \notin \mathcal{A}^*} \alpha_j Z_j]}{\beta_K}\right] \\ &\leq \exp\left(-\tfrac{(\beta_K - 1)^2}{2\beta_K}\right) \ ,\end{aligned}$$

where the first and third inequalities hold since $\mathrm{E}[\sum_{j \notin \mathcal{A}^*} \alpha_j Z_j] \geq \beta_K$, and the second inequality follows from the Chernoff-type bound on the lower tail of the sum of independent $[0,1]$ r.v.s (see, e.g., [1, Thm. 3.5]).

We then fix β_K such that $\exp(-(\beta_K - 1)^2/2\beta_K)$ is at most $1/10K$ (it suffices for β_K to be $O(\log K)$ for this to hold). Consequently, by the union bound, the probability that *some* profit constraint will not be satisfied is at most $1/10$. It follows that our randomized algorithm computes a schedule whose expected cost is $O(\beta_K)\mathsf{Opt}$, and all the profit constraints are met with $\Omega(1)$ probability.