The Complexity of Mean Flow Time Scheduling Problems with Release Times

Philippe Baptiste* Peter Brucker† Marek Chrobak‡ Christoph Dürr*
Svetlana A. Kravchenko§ Francis Sourd¶

February 1, 2008

Abstract

We study the problem of preemptive scheduling n jobs with given release times on m identical parallel machines. The objective is to minimize the average flow time. We show that when all jobs have equal processing times then the problem can be solved in polynomial time using linear programming. Our algorithm can also be applied to the open-shop problem with release times and unit processing times. For the general case (when processing times are arbitrary), we show that the problem is unary NP-hard.

1 Introduction

In the scheduling problem we study, the input instance consists of n jobs with given release times and processing times. The objective is to compute a preemptive schedule of those jobs on m machines that minimizes the average flow time or, equivalently, the sum of completion times, $\sum C_j$. In the standard scheduling notation, the problem can be described as $P|r_j, \text{pmtn}|\sum C_j$.

First, we focus on the case when all jobs have the same processing time p, that is $P|r_j, p_j = p, \text{pmtn}|\sum C_j$. Herrbach and Leung [4] showed that, for $m = 2$, the optimal schedule can be computed in time $O(n \log n)$. For more machines, the complexity of the problem was open. Addressing this open problem, we present an algorithm whose running time is polynomial in m and n.

Our algorithm is based on linear programming. We show that there is always an optimal schedule in a certain *normal* form. We then give a simple linear program of size $O(mn)$, which directly defines an optimal normal schedule. Since the coefficients in the constraints of our linear program are -1, 0, or 1, the result of Tardos [8] implies that the problem can be solved in worst case time $O(n^5 m^5)$.

*CNRS LIX, Ecole Polytechnique, 91128 Palaiseau, France. {baptiste,durr}@lix.polytechnique.fr. Supported by the NSF/CNRS grant 17171 and ANR/Alpage.

†Universität Osnabrück, Fachbereich Mathematik/Informatik, 49069 Osnabrück, Germany. peter@mathematik.uni-osnabrueck.de. Supported by INTAS Project 00-217 and by DAAD PROCOPE Project D/0427360.

‡Department of Computer Science, University of California, Riverside, CA 92521. marek@cs.ucr.edu. Supported by NSF grants CCR-0208856 and INT-0340752.

§United Institute of Informatics Problems, Surganova St. 6, 220012 Minsk, Belarus. kravch@newman.bas-net.by. Supported by the Alexander von Humboldt Foundation.

¶CNRS LIP6, Université Pierre et Marie Curie, Place de Jussieu, 75005 Paris, France. Francis.Sourd@lip6.fr.

We show that, without loss of generality, we can assume that the preemptions occur only at integer times. This yields a polynomial-time algorithm for the open shop problem $O|r_i, p_{ij} = 1|\sum C_i$, for it is known that this problem is equivalent to $P|r_i, p_i = m, \text{pmtn}^+|\sum C_i$, where m is the number of machines [2]. (Notation pmtn^+ means that preemptions are allowed only at integer times.) Previously, it was only known that this problem can be solved in polynomial time if m is constant [9].

In the open shop problem $O|r_i, p_{ij} = 1|\sum C_i$ each job has to be processed on each machine exactly once and with a unit processing time. At any time each machine can execute at most one job and every job can be scheduled by at most one machine. No job can start before its given release time, and the goal is to minimize the total completion time.

In the last section we consider the general case, when the processing times are arbitrary. Du, Leung and Young [3] proved this problem is binary NP-hard for two machines. We show that if the number of machines is not fixed then the problem is in fact unary NP-complete.

We summarize the results discussed above in Table 1.

Problem	Complexity
$P2\|r_j; \text{pmtn}; p_j = p\|\sum C_j$	solvable in time $O(n \log n)$ [4]
$P\ \|r_j; \text{pmtn}; p_j = p\|\sum C_j$	solvable in polynomial time [this paper]
$P2\|r_j; \text{pmtn}\ \|\sum C_j$	binary NP-hard [3]
$P\ \|r_j; \text{pmtn}\ \|\sum C_j$	unary NP-complete [this paper]
$P\ \|\ \text{pmtn}; p_j = p\|\sum C_j$	solvable by the greedy algorithm (trivial)
$P\ \|r_j;\ p_j = p\|\sum C_j$	solvable by the greedy algorithm (trivial)
$1\ \|r_j; \text{pmtn}; p_j = p\|\sum w_j C_j$	open
$P\ \|r_j; \text{pmtn}; p_j = p\|\sum w_j C_j$	unary NP-complete [6]
$O\ \|r_j;\ p_{ji} = 1\|\sum C_j$	solvable in $O(n^2 m^{6m})$ [9], in polynomial time [this paper]

Table 1: Complexity of related scheduling problems. $P2$ stands for the two-machine problem, and O for the open-shop problem. In problems with the objective function $\sum w_j C_j$, each job j is assigned a weight $w_j \geq 0$, and the goal is to minimize the weighted sum of all completion times.

2 Structural Properties

Basic definitions. Throughout the paper, n and m denote, respectively, the number of jobs and the number of machines. The jobs are numbered $1, 2, \ldots, n$ and the machines are numbered $1, 2, \ldots, m$. All jobs have the same length p. For each job j, r_j is the release time of j, where, without loss of generality, we assume that $0 = r_1 \leq \ldots \leq r_n$. In this section we assume that all numbers $p, r_1, \ldots, r_n$ are integers; in the appendix we show that our results can be extended to arbitrary real numbers.

We define a *schedule* $\mathcal{X}$ to be a function which, for any time t, determines the set $\mathcal{X}(t)$ of jobs that are running at time t. This set $\mathcal{X}(t)$ is called the *profile* at time t. Let $\mathcal{X}^{-1}(j)$ denote the set of times when j is executed, that is $\mathcal{X}^{-1}(j) = \{t : j \in \mathcal{X}(t)\}$. In addition we require that $\mathcal{X}$ satisfies the following conditions:

(s1) At most m jobs are executed at any time, that is $|\mathcal{X}(t)| \leq m$ for all times t.

(s2) No job is executed before its release time, that is, for each job j, if $t \quad r_j$ then $j \in \mathcal{X}(t)$.

(s3) Each job runs in a finite number of time intervals. More specifically, for each job j, $\mathcal{X}^{-1}(j)$ is a finite union of intervals of type $[s,t)$.

(s4) Each job is executed for time p, that is $|\mathcal{X}^{-1}(j)| = p$.

In (s4), for a set X of real numbers we use $|X|$ to denote its measure. It is not difficult to see that condition (s3) can be relaxed to allow jobs to be executed in infinitely (but countably) many intervals, without changing the value of the objective function.

By $C_j = \sup \mathcal{X}^{-1}(j)$ we denote the completion time of a job j. In this paper, we are interested in computing a schedule that minimizes the objective function $\sum_{j=1}^{n} C_j$.

Note that, since we are dealing with preemptive schedules, it does not matter to which specific machines the jobs in $\mathcal{X}(t)$ are assigned to. When such an assignment is needed, we will use the convention that the jobs are assigned to machines in the increasing order of indices (or, equivalently, release times): the job with minimum index is assigned to machine 1, the second smallest job to machine 2, etc.

Integral schedules. We say that a schedule $\mathcal{X}$ is *integral*, if $\mathcal{X}(t)$ is constant for $t \in [s, s+1)$ at any $s \in \mathbb{N}$. (In other words, all preemptions occur at integer times.) The following lemma is due to [1], and we include it here for the sake of completeness.

Lemma 1 *There exists an optimal schedule $\mathcal{X}$ that is integral.*

Proof: Let $\mathcal{X}$ be an arbitrary optimal schedule. We will show that there is an integral schedule $\mathcal{X}'$ whose objective value is not greater than that of $\mathcal{X}$. The proof is based on a flow network model of the scheduling problem, and uses the fact that if all capacities are integral then there is an integral solution.

Let $C_1, \ldots, C_n$ be the completion times in $\mathcal{X}$. We consider the set of all time points

$$\{r_1, \ldots, r_n, \lfloor C_1 \rfloor, \lceil C_1 \rceil, \ldots, \lfloor C_n \rfloor, \lceil C_n \rceil\}.$$

Suppose that there are k distinct numbers in this set. We rename these numbers $t_1, ..., t_k$ and order them in increasing order, $t_1 \quad \ldots \quad t_k$. These numbers define $k-1$ intervals $[t_i, t_{i+1})$ for $1 \leq i \quad k$.

We define a network which consists of the nodes $u_1, \ldots, u_n, v_1, \ldots, v_{k-1}$, plus two more nodes designated as the source and the sink. A node u_j represents job j and a node v_i represents interval $[t_i, t_{i+1})$. For every job j, there is an arc from the source to u_j with capacity p and cost 0. For every time interval $[t_i, t_{i+1})$ there is an arc from v_i to the sink with capacity $m(t_{i+1} - t_i)$ and cost 0. For every job j and every interval $[t_i, t_{i+1}) \subseteq [r_j, \lfloor C_j \rfloor)$, there is an arc from u_j to v_i with capacity $t_{i+1} - t_i$ and cost 0. In addition, for every job j for which C_j is not integral, there is an arc from u_j to v_i for $t_i = \lfloor C_j \rfloor$ with capacity 1 and cost 1. (These are the only arcs with non zero cost.)

The schedule $\mathcal{X}$ corresponds to a flow of value np, where the flow on an arc (u_j, v_i) has value $|\mathcal{X}^{-1}(j) \cap [t_i, t_{i+1})|$. The flows on other arcs are uniquely determined by the flows on all arcs $[u_j, v_i)$. The cost of this flow is $w = \sum_{j=1}^{n} (C_j - \lfloor C_j \rfloor)$.

Now we consider the minimum cost flow with maximal value np in this network. Let its cost be $w' \leq w$. This minimum cost flow corresponds to a schedule $\mathcal{X}'$ in the following manner. For each given i, the amount of each job j scheduled in interval $[t_i, t_{i+1})$ is equal to the flow on the

arc (u_j, v_i), which is bounded by the capacity $= t_{i+1} - t_i$ of this arc. The total processing time m in this interval is assigned to jobs $1, 2, \ldots, n$, in this order, processor by processor, and for each processor from left to right. Since each (u_j, v_i) has capacity , a job will not be scheduled at two processors at the same time. Also, since the capacity of the arc between v_i and the sink is m , all jobs will be allocated the required processing time.

Since all arcs have integer capacity, the minimum cost flow can be assumed to be integer (see [7].) Therefore the resulting schedule $\mathcal{X}'$ is integral. It remains to show that its objective value is not larger than that of $\mathcal{X}$.

For each job j, let C'_j be the completion time of j in $\mathcal{X}'$. By the construction of the network, we have $C'_j \leq \lceil C_j \rceil$. Moreover, for w' jobs j we have $C'_j = \lceil C_j \rceil \quad \lfloor C_j \rfloor$, and for these jobs C_j is not integer, while for all the other $n - w'$ jobs j we have $C'_j \leq \lfloor C_j \rfloor$. Setting $\tilde{C} = \sum_{j=1}^{n} \lfloor C_j \rfloor$, the cost of $\mathcal{X}'$ is

$$\sum_{j=1}^{n} C'_j \;\leq\; \tilde{C} + w' \;\leq\; \tilde{C} + w \;\leq\; \tilde{C} + \sum_{j=1}^{n} (C_j - \lfloor C_j \rfloor) \;=\; \sum_{j=1}^{n} C_j,$$

completing the proof of the lemma. □

Busy schedules. We now show that, for the purpose of minimizing our objective function, we can restrict our attention to schedules with some additional properties.

We say that a schedule $\mathcal{X}$ is *busy* if it satisfies the following condition: for any two times $s \quad t$ and a job j such that $r_j \leq s$, if $j \in \mathcal{X}(t)$ and $|\mathcal{X}(s)| \quad m$ (that is, some machine is idle at s) then $j \in \mathcal{X}(s)$ as well. Any optimal schedule is busy, for otherwise, if the above condition is not satisfied, we can move a sufficiently small portion $\quad 0$ of j from the last block where it is executed to the interval $[s, s+\ )$, obtaining a feasible schedule in which the completion time of j is reduced by and other completion times do not change. Thus we only need to be concerned with busy schedules.

Reductions. Let $\mathcal{X}$ be a schedule, and i, j be two jobs with $r_i \quad r_j$. Let T be the set of times where exactly one of the jobs i, j is scheduled, that is $T = [\mathcal{X}^{-1}(i) \setminus \mathcal{X}^{-1}(j)] \cup [\mathcal{X}^{-1}(j) \setminus \mathcal{X}^{-1}(i)]$. Also, let t_0 be a time point which divides T into two parts of the same size, that is $|T \cap [0, t_0)| = \frac{1}{2}|T|$. Since i and j have equal processing time, the processing times of each of i and j in T is equal $\frac{1}{2}|T|$. The (i, j)-*reduction* modifies the schedule by executing i in $|T \cap [0, t_0)|$ and j in $T \cap [t_0, \infty)$. If the reduction does not change $\mathcal{X}$ (that is, i is executed in $|T \cap [0, t_0)|$ and j in $T \cap [t_0, \infty)$) then we say that *i and j are in order*.

We say that a schedule $\mathcal{X}$ is *irreducible* if it is busy and all pairs of jobs are in order. It is not difficult to see that $\mathcal{X}$ is irreducible iff it satisfies the following condition for any times $s \quad t$:

$$\max(\, \mathcal{X}(s) - \mathcal{X}(t) \,) \qquad \min(\, \mathcal{X}(t) - \mathcal{X}(s) \,), \tag{1}$$

where the order refers to job indices, which we assumed to satisfy $i \quad j \Rightarrow r_i \leq r_j$. Also, we use the convention that $\max(\emptyset) = -\infty$ and $\min(\emptyset) = +\infty$, so (1) holds whenever $\mathcal{X}(s) \subseteq \mathcal{X}(t)$ or $\mathcal{X}(t) \subseteq \mathcal{X}(s)$.

Lemma 2 *For any two jobs i, j with $r_i \quad r_j$, an (i, j)-reduction of $\mathcal{X}$ does not increase the objective function, and preserves the integrality and the property of being busy.*

Proof: In the (i,j)-reduction, only the completion times of i and j might change. Let C_i, C_j be the completion times of i, j, before the reduction and C'_i, C'_j after the reduction. We have three cases. If $C_i \quad C_j$ then $C'_j = C_i$ and $C'_i \leq C_j$. If $C_i = C_j$, then the completion times do not change. If $C_i \quad C_j$, then $C'_j = C_j$ and $C'_i \leq C_i$. Thus in all three cases $C_i + C_j$ does not increase.

To justify the second part of the lemma, note that the reduction does not change the cardinality of $\mathcal{X}(t)$ at any time point t. Therefore the new schedule remains busy. Further, if $\mathcal{X}$ is integral then t_0 can be assumed to be an integer, and the new schedule will then be integral as well. □

Theorem 3 *There is an optimal schedule $\mathcal{X}$ that is irreducible.*

Proof: Let $\mathcal{X}$ be an optimal schedule that is busy and integral. We define a potential function which decreases strictly when a reduction of two jobs which are not in order is applied. From this we conclude that after a finite number of reductions we must reach an irreducible schedule.

Define $H(\mathcal{X}) = (H_1(\mathcal{X}), H_2(\mathcal{X}), \ldots, H_n(\mathcal{X}))$, where $H_j(\mathcal{X}) = \sum_{t \in \mathbb{N}, t \in \mathcal{X}^{-1}(j)} t$ for each job j. Given two different schedules $\mathcal{X}$, $\mathcal{Y}$, we say that $H(\mathcal{X})$ is *lexicographically smaller* than $H(\mathcal{Y})$, if $H_i(\mathcal{X}) \quad H_i(\mathcal{Y})$ for the smallest i for which $H_i(\mathcal{X}) \neq H_i(\mathcal{Y})$.

If two jobs $i \quad j$ are not in order, then the (i,j)-reduction decreases $H_i(\mathcal{X})$, while $H_j(\mathcal{X})$ increases. Therefore $H(\mathcal{X})$ decreases lexicographically. Any value $H_j(\mathcal{X})$ is integer and bounded by $0 \leq H_j(\mathcal{X}) \leq p(\max_i C_i)$, where $\max_i C_i$ is preserved by the reductions. From this we can conclude that after a finite number of reductions we obtain an irreducible optimal schedule. □

We now give a characterization of irreducible schedules that will play a major role in the construction of our linear program.

For a given job j and a time t we partition $\mathcal{X}(t) - \{j\}$ into jobs released before and after j. Formally, $\mathcal{X}_{<j}(t) = \{i \in \mathcal{X}(t) : i \quad j\}$ and $\mathcal{X}_{>j}(t) = \{i \in \mathcal{X}(t) : i \quad j\}$. The lemma below provides a characterization of irreducible schedules. (See Figure 1.)

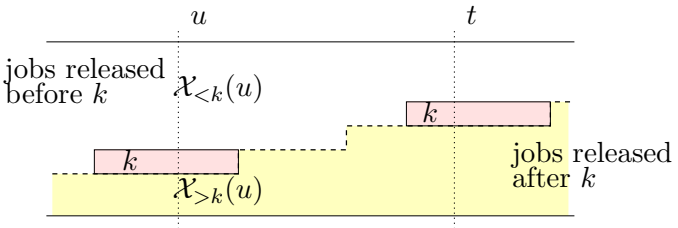

Figure 1: Structure of any irreducible schedule.

Lemma 4 *Let $\mathcal{X}$ be an irreducible schedule. Let u, t be two time points and k be a job such that $r_k \leq u \quad t$. Then:*
(a) *If $k \in \mathcal{X}(u) - \mathcal{X}(t)$ then $|\mathcal{X}_{<k}(t)| \leq |\mathcal{X}_{<k}(u)|$.*
(b) *If $k \in \mathcal{X}(t) - \mathcal{X}(u)$ then $|\mathcal{X}(u)| = m$, $|\mathcal{X}_{>k}(t)| \geq |\mathcal{X}_{>k}(u)|$, and $|\mathcal{X}_{<k}(t)| \quad |\mathcal{X}_{<k}(u)|$.*
(c) *If $k \in \mathcal{X}(u) \cap \mathcal{X}(t)$ then $|\mathcal{X}_{<k}(t)| \leq |\mathcal{X}_{<k}(u)|$ and $|\mathcal{X}_{>k}(t)| \geq |\mathcal{X}_{>k}(u)|$.*

Proof: Case (a) $k \in \mathcal{X}(u) - \mathcal{X}(t)$: We will show by contradiction that $\mathcal{X}_{<k}(t) \subseteq \mathcal{X}_{<k}(u)$, so suppose that there is a job $j \in \mathcal{X}_{<k}(t) - \mathcal{X}_{<k}(u)$. Then $j \geq \min(\mathcal{X}_{<k}(t) - \mathcal{X}_{<k}(u)) = \min(\mathcal{X}(t) - \mathcal{X}(u))$. Also $j \quad k$, and $k \leq \max(\mathcal{X}(u) - \mathcal{X}(t))$. This contradicts irreducibility by equation (1) and thus (a) follows.

Case (b) $k \in \mathcal{X}(t) - \mathcal{X}(u)$: Since $k \notin \mathcal{X}(u)$ and $r_k \leq u$, the assumption that $\mathcal{X}$ is busy implies that $|\mathcal{X}(u)| = m$.

We must have $\mathcal{X}_{>k}(u) \subseteq \mathcal{X}_{>k}(t)$, for otherwise, the existence of $k \in \mathcal{X}(t) - \mathcal{X}(u)$ and an $l \in \mathcal{X}_{>k}(u) - \mathcal{X}_{>k}(t)$ would contradict irreducibility. The inequality $|\mathcal{X}_{>k}(u)| \leq |\mathcal{X}_{>k}(t)|$ follows. This, the assumption of the case, and $|\mathcal{X}(u)| = m$ imply $|\mathcal{X}_{<k}(u)|$ $|\mathcal{X}_{<k}(t)|$.

Case (c) $k \in \mathcal{X}(u) \cap \mathcal{X}(t)$: We only prove the first inequality, as the proof for the second one is very similar. Towards contradiction, suppose $|\mathcal{X}_{<k}(u)|$ $|\mathcal{X}_{<k}(t)|$, and pick any $i \in \mathcal{X}_{<k}(t) - \mathcal{X}_{<k}(u)$. Then $r_i \leq r_k \leq u$ and $i \in \mathcal{X}(t) - \mathcal{X}(u)$, and so the assumption that $\mathcal{X}$ is busy implies $\mathcal{X}(u) = m$. This, in turn, implies that $|\mathcal{X}_{>k}(u)|$ $|\mathcal{X}_{>k}(t)|$, so we can choose $j \in \mathcal{X}_{>k}(u) - \mathcal{X}_{>k}(t)$. But this means that j k i and $j \in \mathcal{X}(u) - \mathcal{X}(t)$, and the existence of such i and j contradicts irreducibility. □

3 A Linear Program for $P|r_j; \text{pmtn}; p_j = p| \sum C_j$

Machine assignment. We now consider the actual job-machine assignment in an irreducible schedule $\mathcal{X}$. As explained earlier, at every time t we assign the jobs in $\mathcal{X}(t)$ to machines in order, that is job $j \in \mathcal{X}(t)$ is assigned to machine $1 + |\mathcal{X}_{<j}(t)|$. Lemma 4 implies that, for any fixed j, starting at $t = r_j$ the value of $|\mathcal{X}_{<j}(t)|$ decreases monotonically with t. Therefore, with machine assignments taken into account, $\mathcal{X}$ will have the structure illustrated in Figure 1.

Call a schedule $\mathcal{X}$ *normal* if for each job j and each machine q, job j is executed on q in a single (possibly empty) interval $[S_{j,q}, C_{j,q})$, and

(1) $C_{j,q} \leq S_{j+1,q}$ for each machine q and job j n, and

(2) $C_{j,q} \leq S_{j,q-1}$ for each machine q 1 and job j.

By the earlier discussion, every irreducible schedule is normal (although the reverse does not hold.) An example of a normal (and irreducible) schedule is shown in Figure 2.

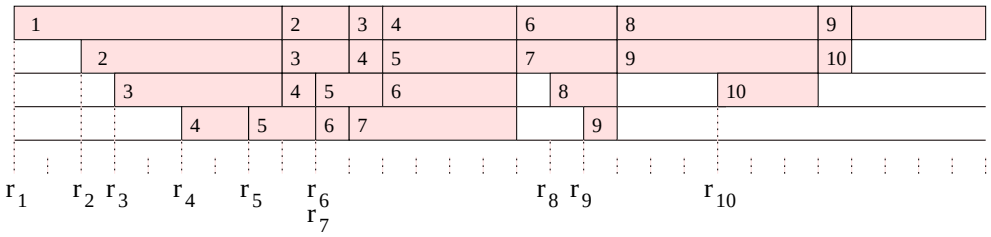

Figure 2: Example of a normal schedule. The processing time is $p = 8$.

Linear program. We are now ready to construct our linear program:

$$
\begin{array}{lrll}
\text{minimize} & \sum_{j=1}^{n} C_{j,1} & & \qquad (2)\\
\text{subject to} & -S_{j,m} & \le -r_j & j = 1, \ldots, n \\
& \sum_q (C_{j,q} - S_{j,q}) & = p & j = 1, \ldots, n \\
& S_{j,q} - C_{j,q} & \le 0 & j = 1, \ldots, n,\ q = 1, \ldots, m \\
& C_{j,q} - S_{j,q-1} & \le 0 & j = 1, \ldots, n,\ q = 2, \ldots, m \\
& C_{j,q} - S_{j+1,q} & \le 0 & j = 1, \ldots, n-1,\ q = 1, \ldots, m
\end{array}
$$

The correspondence between normal schedules and feasible solutions to this linear program should be obvious. For any normal schedule, the start times $S_{j,q}$ and completion times $C_{j,q}$ satisfy the constraints of (2). And vice versa, for any set of the numbers $S_{j,q}$, $C_{j,q}$ that satisfy the constraints of (2), we get a normal schedule by scheduling any job j in interval $[S_{j,q}, C_{j,q})$ on each machine q. Thus we can identify normal schedules $\mathcal{X}$ with feasible solutions of (2). Note, however, that in $\mathcal{X}$ a job j could complete earlier than $C_{j,1}$ (this can happen when $C_{j,1} = S_{j,1}$.) Thus the only remaining issue is whether the optimal normal schedules correspond to optimal solutions of (2).

Theorem 5 *The linear program above correctly computes an optimal schedule. More specifically,* $\min_{\mathcal{X}} \sum_j C_j = \min \sum_j C_{j,1}$, *where on the left-hand side the minimum is over normal schedules* $\mathcal{X}$ *with* C_j *representing the completion time of job* j *in* $\mathcal{X}$, *and on the right-hand side we have the optimal solution of the linear program* (2).

Proof: ($\le$) By the correspondence between normal schedules and feasible solutions of (2), discussed before the theorem, we have $C_j \le C_{j,1}$ for all j, and thus the $\le$ inequality is trivial.

($\ge$) To justify the other inequality, fix an optimal irreducible (and thus also normal) schedule $\mathcal{X}$. We want to show a feasible solution to (2) for which $C_j = C_{j,1}$ for all j.

Fix a job j, and let $[S_{j,g}, C_{j,g})$ be the last (that is, the one with minimum g) non-empty execution interval of j. So $C_j = C_{j,g}$ in $\mathcal{X}$. Consider a block $[s,t)$ where $t = C_{j,g}$. By Lemma 4, all jobs executed on machines $1, 2, \ldots, g-1$ in $[s,t)$ are numbered lower than j. Further, by the ordering of completion times, they are not executed after t. Thus these jobs must be completed at t as well. Therefore we can set $[S_{j,h}, C_{j,h}) = [t,t)$, for all machines $1 \le h \quad g$. This gives a normal schedule in which $C_j = C_{j,1}$.

Having done this for all jobs, we will get numbers $S_{j,q}$ and $C_{j,q}$ that satisfy all constraints of the linear program, and such that $C_j = C_{j,1}$ for all j. □

4 Unary NP-Hardness of $P|r_j, \text{pmtn}|\sum C_j$

In this section we prove that without the assumption on equal processing times the problem is strongly (unary) NP-hard.

Theorem 6 *The problem* $P|r_j, \text{pmtn}|\sum C_j$ *is strongly NP-hard, that is, it is NP-hard even if the numbers on input are represented in the unary encoding.*

Proof: The proof is by reduction from 3-PARTITION. In an instance $\mathcal{I}$ of 3-PARTITION we have $3n+1$ numbers $x_1, x_2, \ldots, x_{3n}, y$ such that $\sum_{i=1}^{3n} x_i = ny$ and $\frac{1}{4}y \quad x_i \quad \frac{1}{2}y$ for each i. We want to determine whether there is a partition of $\{1, 2, \ldots, 3n\}$ into n sets $S_1, \ldots, S_n$ such that $\sum_{i \in S_k} x_i = y$ for all k. (By the assumption about the numbers x_i, in any such partition all sets S_k have exactly three elements.)

Given an instance of 3-PARTITION above, we construct an instance $\mathcal{J}$ of $P|r_j, \text{pmtn}|\sum C_j$ as follows. Define $A = 6ny$ and $B = 18n^2y^2$. We let $m = n$, and $N = 4n + An$. We create three types of jobs:

x-jobs: For each $j = 1, \ldots, 3n$, we create job j with $r_j = 0$ and $p_j = Ax_j$.

B-jobs: For each $j = 3n+1, \ldots, 4n$, we create job j with $r_j = Ay$ and $p_j = B$.

1-jobs: For each $j = 4n+1, \ldots, N$, we create job j with $r_j = Ay + B$ and $p_j = 1$.

If $\mathcal{I}$ is represented in unary, its size is $\Theta(ny)$. Then the size of $\mathcal{J}$ (in unary encoding) is polynomial and the above transformation works in polynomial time.

Thus now it is sufficient to prove the following claim: $\mathcal{I}$ has a 3-partition if and only if $\mathcal{J}$ has a schedule with $\sum_{j=1}^{N} C_j \leq D$, where $D = 3nAy + n(Ay + B) + n\sum_{i=1}^{A}(Ay + B + i)$.

($\Rightarrow$) Let $S_1, \ldots, S_n$ be a 3-partition of $\mathcal{I}$. For each k we have $\sum_{j \in S_k} p_j = Ay$. So on machine k we schedule the following jobs: first the three x-jobs $j \in S_k$, completing the last one at Ay, then one B-job, followed by A 1-jobs. The objective value of this schedule is

$$\sum_{j=1}^{N} C_j \quad \leq \quad 3nAy + n(Ay + B) + n\sum_{i=1}^{A}(Ay + B + i) \;=\; D.$$

($\Leftarrow$) Suppose now that $\mathcal{I}$ does not have a 3-partition. Consider any schedule $\mathcal{X}$ of $\mathcal{J}$ on n machines. We want to prove that the value of the objective function for $\mathcal{X}$ exceeds D.

We consider easy cases first. If some x-job is completed after time $Ay + B$ then

$$\sum_{j=1}^{N} C_j \qquad Ay + B + n(Ay + B) + n\sum_{i=1}^{A}(Ay + B + i) \qquad D.$$

If some B-job is completed after time $Ay + 2B$ then

$$\sum_{j=1}^{N} C_j \qquad (n-1)(Ay + B) + Ay + 2B + n\sum_{i=1}^{A}(Ay + B + i) \;\geq\; D.$$

Suppose now that each x-job is completed no later than at time $Ay+B$ and each B-job is completed no later than at time $Ay + 2B$. Since $\mathcal{I}$ does not have a 3-partition, some machine is idle for A time units in the interval $[0, Ay]$. Therefore some B-job will complete between $Ay + B + A$ and $Ay + 2B$, keeping one machine busy in the interval $[Ay + B, Ay + B + A]$. This implies that there are at least A 1-jobs that will be scheduled at time $Ay + A + B$ or later. Thus

$$\sum_{j=1}^{N} C_j \qquad (n-1)(Ay + B) + Ay + A + B + (n-1)\sum_{i=1}^{A}(Ay + B + i) + A(Ay + A + B) \qquad D.$$

Summarizing, in all cases the objective value exceeds D, completing the proof of the claim. □

5 Final Remarks

We proved that the scheduling problem $P|r_j, \text{pmtn}, p_j = p|\sum C_j$ can be reduced to solving a linear program with $O(mn)$ variables and constraints. This leads to a simple polynomial time algorithm. Since we can assume that $m \leq n$ (otherwise the problem is trivial) the running time can be also expressed as a polynomial of n only.

We showed that there is an optimal schedule with $O(mn)$ preemptions. We do not know whether this bound is asymptotically tight. It is quite possible that there exist optimal schedules in which the number of preemptions is $O(n)$, independent of m. If this is true, this could lead to even more efficient, combinatorial (not dependent on linear programming) algorithms for this problem. Such improvement, however, would require a deeper study of the structural properties of optimal schedules. Since we use only the existence of normal optimal schedules, rather than irreducible schedules, we feel that the problem has more structure to be exploited.

An interesting related open question is $P|r_j, p_j = p|\sum U_j$, where the objective is to find a maximal subset of jobs which can be scheduled on time. It has been shown in [5] that the corresponding preemptive version, $P|r_j, p_j = p, \text{pmtn}|\sum U_j$, is unary NP-hard.

We implemented the complete algorithm (converting the instance to a linear program and solving this linear program). It is accessible at Christoph Dürr's webpage.

References

[1] Ph. Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine when processing times are equal. *Journal of Scheduling*, 2:245–252, 1999.

[2] P. Brucker, B. Jurisch, and M. Jurisch. Open shop problems with unit time operations. *Zeitschrift für Operations Research*, 37:59–73, 1993.

[3] J. Du, J.Y.-T. Leung, and G.H. Young. Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, 75:347–355, 1990.

[4] L.A. Herrbach and J.Y.-T. Leung. Preemptive scheduling of equal length jobs on two machines to minimize mean flow time. *Operations Research*, 38:487–494, 1990.

[5] S. A. Kravchenko. On the complexity of minimizing the number of late jobs in unit time open shop. *Discrete Applied Mathematics*, 100:127–132, 2000.

[6] J.Y.-T. Leung and G.H. Young. Preemptive scheduling to minimize mean weighted flow time. *Information Processing Letters*, 34:47–50, 1990.

[7] A. Schrijver. *Combinatorial Optimization*. Springer Verlag, 2003.

[8] E. Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34:250–256, 1986.

[9] T. Tautenhahn and G. J. Woeginger. Minimizing the total completion time in a unit-time open shop with release times. *Operations Research Letters*, 20:207–212, 1997.

A Proof of Theorem 3 for Arbitrary Real Numbers

In Section 2 we proved that there is always an optimal irreducible schedule, under the assumption that the processing time p and the release times $r_1, \ldots, r_n$ are integral. We now show that this is true in the more general scenario when all these numbers are arbitrary reals.

We first extend the definition of the potential function. For any job j, let $\bar{H}_j(\mathcal{X}) = \frac{1}{2}\int_{\mathcal{X}^{-1}(j)} t\,\mathrm{d}t$, where the integral is taken over the support of j. (We remark that this is a standard value in scheduling, even though the factor $\frac{1}{2}$ is irrelevant for this paper.) The generalized potential function is $\bar{H}(\mathcal{X}) = (\bar{H}_1(\mathcal{X}), \ldots, \bar{H}_n(\mathcal{X}))$.

The following properties remain true: (i) a reduction does not increase the objective function and preserves the property of a schedule being busy, (ii) $\bar{H}(\mathcal{X})$ strictly decreases (lexicographically) after each reduction.

The major difficulty that we need to overcome is that the set of schedules is not closed as a topological space, so there could be a sequence of schedules with decreasing values of $\bar{H}(\mathcal{X})$ whose limit is not a legal schedule. The idea of the proof is to reduce the problem to minimizing $\bar{H}(\mathcal{X})$ over a compact subset of schedules.

Lemma 7 *Even if $p, r_1, \ldots, r_n$ are arbitrary real numbers, there is an irreducible optimal schedule.*

Proof: Define a *block* of a schedule $\mathcal{X}$ to be a maximal time interval $[u, t)$ such that (u, t) does not contain any release times and $\mathcal{X}(s)$ is constant for $s \in [u, t)$.

For convenience, let r_{n+1} be any upper bound on the last completion time of any optimal schedule, say $r_{n+1} = r_n + np$. Thus all jobs are executed between r_1 and r_{n+1}. Each interval $[r_i, r_{i+1})$, for $i = 1, \ldots, n$ is called a *segment*. By condition (s3), each segment is a disjoint union of a finite number of blocks of $\mathcal{X}$. Also, for each job j, we have $C_j = t$ for the last non-empty block $[s, t)$ whose profile contains j.

A schedule $\mathcal{X}$ is called *tidy* if all jobs are completed no later than at r_{n+1} and, for any segment $[r_i, r_{i+1})$, the profiles $\mathcal{X}(t)$, for $t \in [r_i, r_{i+1})$, are lexicographically ordered from left to right. More precisely, this means that, for any $r_i \leq s \quad t \quad r_{i+1}$, we have

$$\min(\,\mathcal{X}(s) - \mathcal{X}(t)\,) \quad \leq \quad \min(\,\mathcal{X}(t) - \mathcal{X}(s)\,).$$

One useful property of tidy schedules is that its total number of blocks (including the empty ones) is $N = nM$, where $M = \sum_{l=0}^{m} \binom{n}{l}$, because n is the number of intervals $[r_i, r_{i+1})$ and M is the number of lexicographically ordered blocks in each interval. From now on we identify any tidy schedule $\mathcal{X}$ with the vector $\mathcal{X} \in \mathbb{R}^N$ whose b-th coordinate x_b represents the length of the b-th block in $\mathcal{X}$.

In fact, the set $\mathbf{T}$ of tidy schedules is a (compact) convex polyhedron in $\mathbb{R}^N$, for we can describe $\mathbf{T}$ with a set of linear inequalities that express the following constraints:

- Each job j is not executed before r_j,
- Each job j is executed for time p.

For example, the second constraint can be written as $\sum_b x_b = p$, where the sum is over all blocks b whose profile contains j.

Claim 8 *Any schedule $\mathcal{X}$ can be transformed into a schedule with completion times ordered as $C_1 \leq \ldots \leq C_n$, without increasing the objective function value.*

Proof: Note that after an (i,j)-reduction we have $C_i \leq C_j$, and no other completion time is changed. Therefore after reducing job 1 successively with the jobs $2,\ldots,n$, C_1 is the smallest completion time. Then after reducing job 2 with the jobs $3,\ldots,n$ we have $C_1 \leq C_2 \leq \min\{C_3,\ldots,C_n\}$. Continuing this process will eventually end with ordered completion times. □

Claim 9 *Let $\mathcal{X}$ be a schedule in which completion times are ordered and upper bounded by r_{n+1}. Then $\mathcal{X}$ can be converted into a tidy schedule $\mathcal{X}'$ such that*
(a) *$C'_j \leq C_j$ for all j (where C_j and C'_j are the completion times of j in $\mathcal{X}$ and $\mathcal{X}'$, respectively.)*
(b) *$\bar{H}(\mathcal{X}')$ is equal to or lexicographically smaller than $\bar{H}(\mathcal{X})$.*

Proof: Indeed, suppose that $\mathcal{X}$ has two consecutive blocks $A = [u,s)$, $B = [s,t)$ where $r_i \leq u \quad s \quad t \leq r_{i+1}$, and the profile $\mathcal{X}(u)$ of A is larger (lexicographically) than the profile $\mathcal{X}(s)$ of B. Exchange A and B, and denote by $\mathcal{X}'$ the resulting schedule. Let $j = \min(\mathcal{X}(s) - \mathcal{X}(u))$ $\min(\mathcal{X}(u) - \mathcal{X}(s))$. Since $C_j \geq t$, all jobs in $\mathcal{X}(u) - \mathcal{X}(s)$ are also completed not earlier than at t. So this exchange does not increase any completion times. We have $\bar{H}_j(\mathcal{X}') \quad \bar{H}_j(\mathcal{X})$ and $\bar{H}_i(\mathcal{X}') = \bar{H}_i(\mathcal{X})$ for $i \quad j$. Thus $\bar{H}(\mathcal{X}')$ is lexicographically smaller than $\bar{H}(\mathcal{X})$. By repeating this process, we eventually convert $\mathcal{X}$ into a tidy schedule that satisfies the claim. □

We now continue the proof of the lemma. Fix some optimal schedule $\mathcal{X}^*$. Let C^*_j denote the completion time of a job j in $\mathcal{X}^*$. From Claims 8 and 9, we can assume that $\mathcal{X}^*$ is tidy and $C^*_1 \leq C^*_2 \leq \ldots \leq C^*_n$. (For the peace of mind, it is worth noting that Claim 9 implies that $\mathcal{X}^*$ is well defined, for it reduces the problem to minimizing $\sum_j C_j$ over a compact subset $\mathbf{T}$ of $\mathbb{R}^N$.)

Consider a class $\mathbf{T}_0 \subseteq \mathbf{T}$ of tidy schedules $\mathcal{X}$ such that each job j in $\mathcal{X}$ is completed not later than at time C^*_j. Since $\mathcal{X}^* \in \mathbf{T}_0$, the set $\mathbf{T}_0$ is not empty. Similarly as $\mathbf{T}$, $\mathbf{T}_0$ is a (compact) convex polyhedron. Indeed, we obtain $\mathbf{T}_0$ by using the same constraints as for $\mathbf{T}$ and adding the constraints that each job j is completed not later than at C^*_j. To express this constraint, if in $\mathcal{X}^*$ the completion time C^*_j of j is at the end of the a-th block in the segment $[r_i, r_{i+1})$, then for each $b \quad (i-1)M + a$ such that j is in the profile of the b-th block, we would have a constraint $x_b = 0$. Note that these constraints do not explicitly force j to end *exactly* at C^*_j, but the optimality of $\mathcal{X}^*$ guarantees that it will have to.

Now we show that there exists a schedule $\mathcal{Y}^* \in \mathbf{T}_0$ for which $\bar{H}(\mathcal{Y}^*)$ is lexicographically minimum. First, as we explained earlier, $\mathbf{T}_0$ is a compact convex polyhedron. Let $\mathbf{T}_1 \subseteq \mathbf{T}_0$ be the set of $\mathcal{X}$ for which $\bar{H}_1(\mathcal{X})$ is minimized. $\bar{H}_1$ is a continuous quadratic function over $\mathbf{T}_0$, and thus $\mathbf{T}_1$ is also a non-empty compact set. Continuing this process, we construct sets $\mathbf{T}_2,\ldots,\mathbf{T}_n$, and we choose $\mathcal{Y}^*$ arbitrarily from $\mathbf{T}_n$. □