Minimizing the Number of Tardy Jobs with Uniform Processing Times on Parallel Machines

Klaus Heeger
Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Hendrik Molter
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

In this work, we study the computational (parameterized) complexity of $P \mid r_j, p_j = p \mid \sum w_j U_j$. Here, we are given m identical parallel machines and n jobs with equal processing time, each characterized by a release date, a due date, and a weight. The task is to find a feasible schedule, that is, an assignment of the jobs to starting times on machines, such that no job starts before its release date and no machine processes several jobs at the same time, that minimizes the weighted number of tardy jobs. A job is considered tardy if it finishes after its due date.

Our main contribution is showing that $P \mid r_j, p_j = p \mid \sum U_j$ (the unweighted version of the problem) is NP-hard and W[2]-hard when parameterized by the number of machines. The former resolves an open problem in Note 2.1.19 by Kravchenko and Werner [Journal of Scheduling, 2011] and Open Problem 2 by Sgall [ESA, 2012], and the latter resolves Open Problem 7 by Mnich and van Bevern [Computers & Operations Research, 2018]. Furthermore, our result shows that the known XP-algorithm by Baptiste et al. [4OR, 2004] for $P \mid r_j, p_j = p \mid \sum w_j U_j$ parameterized by the number of machines is optimal from a classification standpoint.

On the algorithmic side, we provide alternative running time bounds for the above-mentioned known XP-algorithm. Our analysis shows that $P \mid r_j, p_j = p \mid \sum w_j U_j$ is contained in XP when parameterized by the processing time, and that it is contained in FPT when parameterized by the combination of the number of machines and the processing time. Finally, we give an FPT-algorithm for $P \mid r_j, p_j = p \mid \sum w_j U_j$ parameterized by the number of release dates or the number of due dates. With this work, we lay out the foundation for a systematic study of the parameterized complexity of $P \mid r_j, p_j = p \mid \sum w_j U_j$.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Discrete mathematics; Computing methodologies → Planning and scheduling

Keywords and phrases Scheduling, Identical Parallel Machines, Weighted Number of Tardy Jobs, Uniform Processing Times, Release Dates, NP-hard Problems, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2025.47

Funding *Klaus Heeger*: Supported by the ISF, grant No. 1070/20.
Hendrik Molter: Supported by the ISF, grant nr. 1470/24 and by the European Union's Horizon Europe research and innovation program under grant agreement 949707.

1 Introduction

Machine scheduling is one of the most fundamental application areas of combinatorial optimization [34]. In a typical scheduling problem, the task is to assign jobs to machines with the goal of maximizing a certain optimization objective while complying with certain constraints. Jobs are usually characterized by a *processing time*, a *release date*, a *due date*, and a *weight* (or a subset thereof). We consider the setting where we have access to several identical parallel machines that can each process one job (non-preemptively) at a time. One of the most fundamental optimization objectives is to minimize the weighted number of tardy jobs, where a job is considered *tardy* if it is completed after its due date.

The arguably simplest scheduling problem aims to minimize the (unweighted) number of tardy jobs on a single machine, where all jobs are released at time zero. In the standard three-field notation for scheduling problems by Graham et al. [15], this problem is called $1 \mid\mid \sum U_j$. It can be solved in polynomial time by a classic algorithm of Moore [33]. However, this problem becomes NP-hard when weights are introduced, the number of machines is increased, or release dates are added.

- The weighted version, $1 \mid\mid \sum w_j U_j$, is one of the first scheduling problems shown to be (weakly) NP-hard. It remains hard even if all jobs have the same due date, as in this case, it is equivalent to the well-known KNAPSACK problem, which was already included in Karp's famous list of 21 NP-hard problems [21]. The problem can be solved in pseudopolynomial time with a classic algorithm by Lawler and Moore [28].
- Adding a second machine leads to the problem $2 \mid\mid \sum_j U_j$, which is (weakly) NP-hard even if all jobs have the same due date, as it is a generalization of the well-known PARTITION problem, which was also already included in Karp's list of 21 NP-hard problems [21]. If the number of machines is unbounded, the problem is called $P \mid\mid \sum_j U_j$ and it is strongly NP-hard even if all jobs have the same due date, as it generalizes the well-known BIN PACKING problem [14].
- Introducing release times leads to the problem $1 \mid r_j \mid \sum_j U_j$, which is weakly NP-hard [30], even if there are only two different release dates and two different due dates. The reduction of Lenstra et al. [30] can be extended in a straightforward way to show that $1 \mid r_j \mid \sum U_j$ is strongly NP-hard.

Problem Setting and Motivation. We consider the case where release dates and weights are present and we have multiple identical parallel machines. However, we add the restriction that all processing times are the same. This problem is called $P \mid r_j, p_j = p \mid \sum w_j U_j$, we give a formal definition in Section 2. This problem arises naturally in manufacturing systems, where

- exact specifications by the customers for the product have negligible influence on the production time, but
- the specifications only become available at certain times and customers request the product to meet certain due dates.

As an illustrative example, consider the problem of scheduling burn-in operations in integrated circuit manufacturing. The specification of the layout of the circuit may only become available at a certain time, as it takes time to optimize it. At the same time, the specific layout has little to no influence on the processing time of the burn-in operation [31]. Furthermore, the customer may wish to have the circuit delivered at a certain due date.

To the best of our knowledge, the only known algorithmic result for $P \mid r_j, p_j = p \mid \sum w_j U_j$ is a polynomial-time algorithm by Baptiste et al. [2, 3] for the case where the number of machines is a constant. However, two special cases are known to be polynomial-time solvable. It is folklore that the case without release dates, $P \mid p_j = p \mid \sum_j w_j U_j$, and the case where the processing times equal one, $P \mid r_j, p_j = 1 \mid \sum_j w_j U_j$, can both be reduced to the LINEAR ASSIGNMENT problem in a straightforward manner. The LINEAR ASSIGNMENT problem is known to be solvable in polynomial time [27]. Furthermore, it is known that, given an instance of $P \mid r_j, p_j = p \mid \sum w_j U_j$, we can check in polynomial time whether all jobs can be scheduled such that *no* job is tardy [5, 37, 38].

Our Contribution. The complexity status of $P \mid r_j, p_j = p \mid \sum w_j U_j$ and its unweighted version $P \mid r_j, p_j = p \mid \sum U_j$ was a longstanding open problem. Kravchenko and Werner [25] pointed out that this question remains unanswered in Note 2.1.19 and Sgall [36] listed this issue as Open Problem 2. Our main contribution is to resolve the complexity status of $P \mid r_j, p_j = p \mid \sum U_j$ (and hence also $P \mid r_j, p_j = p \mid \sum w_j U_j$) by showing the following.

- $P \mid r_j, p_j = p \mid \sum U_j$ is NP-hard.

Having established NP-hardness, we focus on studying the parameterized complexity of $P \mid r_j, p_j = p \mid \sum w_j U_j$. As mentioned before, Baptiste et al. [2, 3] showed that the problem is in XP when parameterized by the number of machines. Mnich and van Bevern [32, Open Problem 7] asked whether this result can be improved to an FPT algorithm. We answer this question negatively by showing the following.

- $P \mid r_j, p_j = p \mid \sum U_j$ is W[2]-hard when parameterized by the number of machines.

On the positive side, we give several new parameterized tractability results. By providing an alternative running time analysis of the algorithm for $P \mid r_j, p_j = p \mid \sum w_j U_j$ by Baptiste et al. [2, 3], we show the following.

- $P \mid r_j, p_j = p \mid \sum w_j U_j$ is in XP when parameterized by the processing time.
- $P \mid r_j, p_j = p \mid \sum w_j U_j$ is in FPT when parameterized by the combination of the number of machines and the processing time.

Finally, we give a new algorithm based on a mixed integer linear program (MILP) formulation for $P \mid r_j, p_j = p \mid \sum w_j U_j$. We show the following.

- $P \mid r_j, p_j = p \mid \sum w_j U_j$ is in FPT when parameterized by the number of different release dates.
- $P \mid r_j, p_j = p \mid \sum w_j U_j$ is in FPT when parameterized by the number of different due dates.

We conclude by pointing to new future research directions. Most prominently, we leave open whether $P \mid r_j, p_j = p \mid \sum w_j U_j$ is in FPT or W[1]-hard when parameterized by the processing time.

Related Work. We give an overview of the literature investigating the parameterized complexity of minimizing the weighted number of tardy jobs in various related settings.

The problem of minimizing the weighted number of tardy jobs on a single machine, $1 \mid\mid \sum_j w_j U_j$ has been extensively studied in the literature under various aspects and constraints. Hermelin et al. [20] showed that the classical pseudopolynomial time algorithm by Lawler and Moore [28] can be improved in several special cases. Hermelin et al. [19] give an overview of the parameterized complexity of $1 \mid\mid \sum_j w_j U_j$ with respect to the parameters "number of processing times", "number of due dates", and "number of weights" (and their combinations). In particular, $1 \mid\mid \sum w_j U_j$ is in XP when parameterized by the number of different processing times [19]. This presumably cannot be improved to an FPT result as recently, it was shown that $1 \mid\mid \sum w_j U_j$ parameterized by the number of different processing times is W[1]-hard [16]. Faster algorithms are known for the case where the job weights equal the processing times [4, 12, 23, 35] and the problem has also been studied under fairness aspects [17]. Kaul et al. [22] extended the results of Hermelin et al [19] to $1 \mid r_j \mid \sum w_j U_j$, considering the "number of release times" as an additional parameter.

Minimizing the weighted number of tardy jobs on parallel machines has mostly been studied in the context of interval scheduling ($P \mid d_j - r_j = p_j \mid \sum_j w_j U_j$) and generalizations thereof [1, 18, 26, 39].

The setting where processing times are assumed to be uniform has been studied under various optimization criteria (different from minimizing the weighted number of tardy jobs) and constraints. For an overview, we refer to Kravchenko and Werner [25] and Baptiste et al. [3]. The even more special case of unit processing times has also been extensively studied. Reviewing the related literature in this setting is out of scope for this work.

2 Preliminaries

Scheduling. Using the standard three-field notation for scheduling problems by Graham et al. [15], the problem considered in this work is called $P \mid r_j, p_j = p \mid \sum w_j U_j$. In this problem, we have n jobs and m machines. Each machine can process one job at a time. Generally, we use the variable j to denote jobs and the variable i to denote machines. Each job j has a *processing time* $p_j = p$, a *release date* r_j, a *due date* d_j, and a *weight* w_j, where p, r_j, d_j, and w_j are nonnegative integers. We use $r_\#$, $d_\#$, and $w_\#$ to denote the number of different release dates, due dates, and weights, respectively.

A schedule maps each job j to a combination of a machine i and a starting time t, indicating that j shall be processed on machine i starting at time t. More formally, a *schedule* is a function $\sigma : \{1, \ldots, n\} \to \{1, \ldots, m\} \times \mathbb{N}$. If for job j we have $\sigma(j) = (i, t)$, then job j is scheduled to be processed on machine i starting at time t until time $t + p$. A schedule σ is *feasible* if there is no job j with $\sigma(j) = (i, t)$ and $t < r_j$ and if there is no pair of jobs j, j' with $\sigma(j) = (i, t)$ and $\sigma(j') = (i, t')$ such that $|t - t'| < p$. We say that a job j is *early* in a feasible schedule σ if $\sigma(j) = (i, t)$ and $t + p \leq d_j$, otherwise we say that job j is *tardy*. We say that machine i is *idle* at time t in a feasible schedule σ if there is no job j with $\sigma(j) = (i, t')$ and $t' \leq t \leq t' + p$. The goal is to find a feasible schedule that minimizes the weighted number of tardy jobs or, equivalently, maximizes the weighted number of early jobs $W = \sum_{j \mid \sigma(j)=(i,t) \wedge t+p \leq d_j} w_j$. We call a feasible schedule that maximizes the weighted number of early jobs *optimal*. Formally, the problem is defined as follows.

$P \mid r_j, p_j = p \mid \sum w_j U_j$

Input: A number n of jobs, a number m of machines, a processing time p, a list of release dates $(r_1, r_2, \ldots, r_n)$, a list of due dates $(d_1, d_2, \ldots, d_n)$, and a list of weights $(w_1, w_2, \ldots, w_n)$.

Task: Compute a feasible schedule σ that maximizes $W = \sum_{j \mid \sigma(j)=(i,t) \wedge t+p \leq d_j} w_j$.

We use $P \mid r_j, p_j = p \mid \sum U_j$ to denote the unweighted (or, equivalently, uniformly weighted) version of $P \mid r_j, p_j = p \mid \sum w_j U_j$, that is, the case where $w_j = w_{j'}$ for every two jobs j and j', or equivalently, $w_\# = 1$.

Note that given any feasible schedule of the early jobs, one can easily extend this to a feasible schedule of all jobs as tardy jobs can be scheduled arbitrarily late. Thus, when describing a schedule, we will only describe how the early jobs are scheduled.

Given an instance I of $P \mid r_j, p_j = p \mid \sum w_j U_j$, we can make the following observation, essentially implying that one can switch the roles of release dates and due dates.

▶ **Observation 1.** *Let I be an instance of $P \mid r_j, p_j = p \mid \sum w_j U_j$ and let $d_{\max}$ be the largest due date of any job in I. Let I' be the instance obtained from I by setting $r'_j = d_{\max} - d_j$ and $d'_j = d_{\max} - r_j$. Then I admits a feasible schedule where the weighted number of early jobs is W if and only if I' admits a feasible schedule where the weighted number of early jobs is W.*

To see that Observation 1 is true note that a feasible schedule σ for I can be transformed into a feasible schedule σ' for I' (with the same weighted number of early jobs) by setting $\sigma'(j) = (i, d_{\max} - t - p)$, where $\sigma(j) = (i, t)$.

We now show that we can restrict ourselves to schedules where jobs may start only at "few" different points in time, which will be useful in our proofs. In order to do so, we define a set $\mathcal{T}$ of *relevant starting time points*.

$$\mathcal{T} = \{t \mid \exists\, r_j \text{ and } \exists\, 0 \leq \ell \leq n \text{ s.t. } t = r_j + p \cdot \ell\}$$

It is known that there always exists an optimal schedule where the starting times of all jobs are in $\mathcal{T}$.

▶ **Lemma 2** ([2, 3]). *Let I be an instance of $P \mid r_j, p_j = p \mid \sum w_j U_j$. Then there exists a feasible schedule σ that maximizes the weighted number of early jobs such that for each job j we have $\sigma(j) = (i, t)$ for some $t \in \mathcal{T}$.*

Parameterized Complexity. We use the following standard concepts from parameterized complexity theory [7, 11, 13]. A *parameterized problem* $L \subseteq \Sigma^* \times \mathbb{N}$ is a subset of all instances (x, k) from $\Sigma^* \times \mathbb{N}$, where k denotes the *parameter*. A parameterized problem L is in the class FPT (or *fixed-parameter tractable*) if there is an algorithm that decides every instance (x, k) for L in $f(k) \cdot |x|^{O(1)}$ time for some computable function f that depends only on the parameter. A parameterized problem L is in the class XP if there is an algorithm that decides every instance (x, k) for L in $|x|^{f(k)}$ time for some computable function f that depends only on the parameter. If a parameterized problem L is W[1]-hard or W[2]-hard, then it is presumably not contained in FPT [7, 11, 13].

3 Hardness of $P \mid r_j, p_j = p \mid \sum U_j$

In this section, we present our main result, namely that the unweighted version of our scheduling problem, $P \mid r_j, p_j = p \mid \sum U_j$, is NP-hard and W[2]-hard when parameterized by the number m of machines. The former resolves an open problem in Note 2.1.19 by Kravchenko and Werner [25] and Open Problem 2 by Sgall [36], and the latter resolves Open Problem 7 by Mnich and van Bevern [32].

▶ **Theorem 3.** *$P \mid r_j, p_j = p \mid \sum U_j$ is NP-hard and W[2]-hard parameterized by the number m of machines.*

In order to show Theorem 3, we present a parameterized reduction from Hitting Set parameterized by solution size k, which is known to be NP-hard [21] (unparameterized) and W[2]-hard [10].

Hitting Set

Input: A finite set $U = \{u_0, \ldots, u_{n-1}\}$, a set $\mathcal{A} = \{A_0, \ldots, A_{m-1}\}$ of subsets of U, and an integer k.

Question: Is there a *hitting set* of size k, that is, a set $X \subseteq U$ with $|X| = k$ and $X \cap A_j \neq \emptyset$ for every $j \in \{0, \ldots, m-1\}$?

Let $I = (U = \{u_0, \ldots, u_{n-1}\}, \mathcal{A} = \{A_0, \ldots, A_{m-1}\}, k)$ be an instance of Hitting Set. In order to ease the presentation, we give jobs names rather than identifying them with natural numbers. Furthermore, for a job J, we use $r(J)$ to denote the release date of J, and we use $d(J)$ to denote the deadline of J.

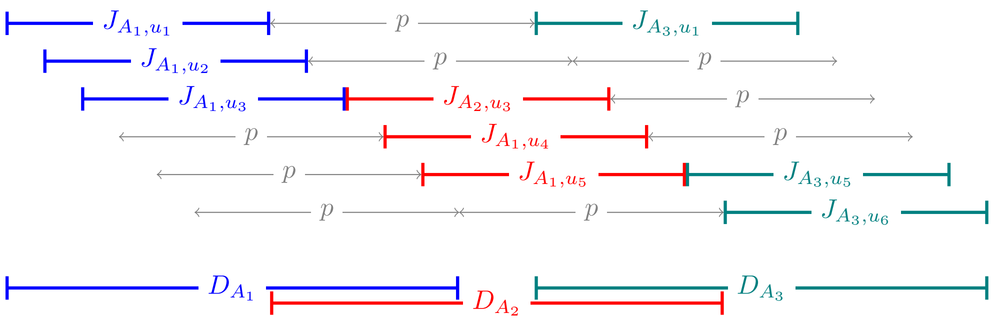

Figure 1 The jobs of the first reduction approach for the Hitting Set instance $I = (\{u_1, u_2, u_3, u_4, u_5, u_6\}, \{A_1 = \{u_1, u_2, u_3\}, A_2 = \{u_3, u_4, u_5\}, A_3 = \{u_1, u_5, u_6\}, k = 2)$.

Our reduction will have k machines. The main idea behind the reduction is as follows. Each machine acts as a "selection gadget", that is, we will interpret the jobs scheduled on each machine in an optimal schedule as the selection of a particular element of U to be included in the hitting set. As there are k machines, this ensures that the (hitting) set consisting of the selected elements has size at most k. Intuitively, we want that selecting element u_i on a machine corresponds to all jobs on this machine starting at time i modulo p in an optimal schedule. For each set $A_j \in \mathcal{A}$, there are two kinds of jobs.

- First, jobs J_{A_j,u_i} for each $u_i \in A_j$, where scheduling job J_{A_j,u_i} encodes that the element u_i is selected to be part of the hitting set.
- Second, there are $k-1$ dummy jobs D_{A_j} which can be scheduled on the up to $k-1$ machines not corresponding to elements of A_j.

We give the jobs J_{A_j,u_i} and D_{A_j} release dates and due dates such that they are the only jobs that can be started in the interval $[j \cdot p, (j+1) \cdot p - 1]$ and are early. See Figure 1 for an illustration. Intuitively, this makes sure that an optimal solution has to schedule one of these jobs on each machine. In particular, one job J_{A_j,u_i} is scheduled, implying that A_j is hit by one of the selected elements. See Figure 2 for an illustration.

There is, however, one problem with the reduction as sketched above. We do not ensure that all early jobs scheduled on some machine start at the same time modulo p. Thus, it is possible to schedule e.g. first job J_{A_1,u_2} on machine 1 and then job J_{A_2,u_3} such that the two jobs are both early. Machine 1 now does not encode the selection of a single element to the hitting set. We illustrate an example in Figure 3. However, note that it is only possible to increase the index of the "selected" element, and as there are only k machines, the total increase is bounded by $k \cdot (n-1)$. Consequently, repeating the reduction sketched above $k \cdot (n-1) + 1$ times ensures that at least one of the repeated instances will select only one item per machine, and then this instance correctly encodes a hitting set of size k.

We now describe the reduction in detail. Formally, given the instance I of Hitting Set, we construct an instance I' of $P \mid r_j, p_j = p \mid \sum U_j$ as follows. We set the processing time to $p = 2n$. We construct the following jobs for each $A_j \in \mathcal{A}$ and $\ell \in \{0, \ldots, k \cdot (n-1)\}$:

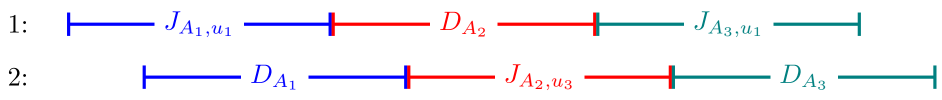

Figure 2 An optimal schedule for the instance from Figure 1 representing the hitting set $\{u_1, u_3\}$.

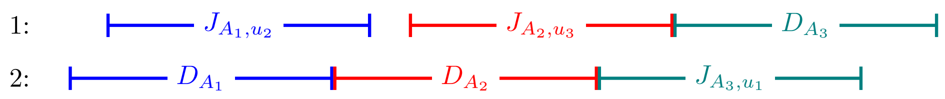

Figure 3 An optimal schedule for the instance from Figure 1 which does not represent a hitting set.

- one job $J^{\ell}_{A_j,u_i}$ for each $u_i \in A_j$ with release date $r(J^{\ell}_{A_j,u_i}) = (\ell \cdot m + j) \cdot p + i$ and due date $d(J^{\ell}_{A_j,u_i}) = r(J^{\ell}_{A_j,u_i}) + p$, and
- $k-1$ dummy jobs $D^{\ell}_{A_j}$ with release date $r(D^{\ell}_{A_j}) = (\ell \cdot m + j) \cdot p$ and due date $d(D^{\ell}_{A_j}) = r(D^{\ell}_{A_j}) + p + n$.

Finally, we set the number of machines to k. This finishes the construction. For an overview of the due dates and release dates of the jobs, see Table 1. We can easily observe the following.

▶ **Observation 4.** *Given an instance I of Hitting Set, the above-described instance I' of $P \mid r_j, p_j = p \mid \sum U_j$ can be computed in polynomial time and has k machines.*

We continue by showing the correctness of the reduction. More specifically, we show that the Hitting Set I instance is a yes-instance if and only if the constructed instance I' of $P \mid r_j, p_j = p \mid \sum U_j$ admits a feasible schedule with $(k \cdot (n-1)+1) \cdot m \cdot k$ early jobs. We split the proof into the forward and backward direction. We start with the forward direction.

▶ **Lemma 5.** *If the Hitting Set instance I admits a hitting set of size k, then the $P \mid r_j, p_j = p \mid \sum U_j$ instance I' admits a feasible schedule with $(k \cdot (n-1)+1) \cdot m \cdot k$ early jobs.*

Proof. Let $X = \{u_{i_1}, \dots, u_{i_k}\}$ be a hitting set of size k for I. We construct a schedule for I' as follows. On machine q, for each $\ell \in \{0, \dots, k \cdot (n-1)\}$ and $j \in \{0, \dots, m-1\}$, we schedule one job to start at time $(\ell \cdot m + j) \cdot p + i_q$. This job is $J^{\ell}_{A_j,u_{i_q}}$ if $u_{i_q} \in A_j$, and $D^{\ell}_{A_j}$ otherwise. Because X is a hitting set, we have that $u_{i_q} \in A_j$ for some $q \in \{1, \dots, k\}$ and hence we schedule each dummy job $D^{\ell}_{A_j}$ at most $k-1$ times, that is, at most its multiplicity times.

We can observe that all jobs scheduled so far are early. For each job $J^{\ell}_{A_j,u_{i_q}}$ that is scheduled on machine q, we have set its starting time to $(\ell \cdot m + j) \cdot p + i_q$ which equals this job's release date (cf. Table 1). Furthermore, job $J^{\ell}_{A_j,u_{i_q}}$ finishes at $(\ell \cdot m + j + 1) \cdot p + i_q$, its deadline. Each dummy job $D^{\ell}_{A_j}$ is early as well, since their release times are smaller or equal to the release time of $J^{\ell}_{A_j,u_{i_q}}$, and their due dates are larger or equal to the due date of $J^{\ell}_{A_j,u_{i_q}}$. Furthermore, we can observe that there is no overlap in the processing times between any two jobs scheduled on machine q.

It follows that we have feasibly scheduled $(k \cdot (n-1)+1) \cdot m \cdot k$ such that they finish early. We schedule the remaining jobs in some arbitrary way such that the schedule remains feasible. ◀

Table 1 Overview of the release dates and due dates of the jobs created for each $A_j \in \mathcal{A}$ and $\ell \in \{0, \dots, k \cdot (n-1)\}$.

job	release date	due date	multiplicity
$J^{\ell}_{A_j,u_i}$	$(\ell \cdot m + j) \cdot p + i$	$(\ell \cdot m + j + 1) \cdot p + i$	1
$D^{\ell}_{A_j}$	$(\ell \cdot m + j) \cdot p$	$(\ell \cdot m + j + 1) \cdot p + n$	$k-1$

Next, we continue with the backward direction.

▶ **Lemma 6.** *If the $P \mid r_j, p_j = p \mid \sum U_j$ instance I' admits a feasible schedule with $(k \cdot (n-1)+1) \cdot m \cdot k$ early jobs, then the* Hitting Set *instance I admits a hitting set of size k.*

Proof. Let σ be a feasible schedule for I' with $(k \cdot (n-1)+1) \cdot m \cdot k$ early jobs. Since the largest due date is $(k \cdot (n-1) \cdot m + m) \cdot p + n$ and $p = 2n$, it follows that on each machine, at most $k \cdot (n-1) \cdot m + m$ jobs can be scheduled in a feasible way such that they finish early. Consequently, on each machine, there are exactly $(k \cdot (n-1)+1) \cdot m$ early jobs.

More specifically, for each machine, each $\ell \in \{0, \dots, k \cdot (n-1)\}$ and $j \in \{0, \dots, m-1\}$, there must be one job which starts in the interval $[(\ell \cdot m + j) \cdot p, (\ell \cdot m + j) \cdot p + n]$. Otherwise, there would be less than $(k \cdot (n-1)+1) \cdot m$ early jobs on that machine. For machine q, let $x_\ell^q \in \{1, \dots, n\}$ such that there is a job on machine q which starts at time $\ell \cdot m \cdot p + x_\ell^q$ (the existence of such a job is guaranteed by the previous observation). Then we must have $1 \le x_0^q \le x_1^q \le \dots \le x_{k\cdot(n-1)}^q \le n$. This follows from the observation that if $x_\ell^q > x_{\ell+1}^q$, then the starting time of the job corresponding to $x_{\ell+1}^q$ would be earlier than the completion time of the job corresponding to x_ℓ^q and hence the schedule would be infeasible. Consequently, there are at most $n-1$ values of ℓ such that $x_\ell^q \neq x_{\ell+1}^q$. As there are k machines, this implies that there are at most $k \cdot (n-1)$ values such that $x_\ell^q \neq x_{\ell+1}^q$ for *some* machine q. We can conclude that there exists at least one $\ell \in \{0, \dots, k \cdot (n-1)\}$ so that $x_\ell^q = x_{\ell+1}^q$ for every machine q. This implies that for each $j \in \{0, \dots, m-1\}$ and each machine q, there is one job starting at time $(\ell \cdot m + j) \cdot p + x_\ell^q$ on machine q. We fix such an ℓ for the rest of the proof and claim that $X = \{u_{x_\ell^1}, \dots, u_{x_\ell^k}\}$ is a hitting set of size at most k for I.

Clearly, X has size at most k. Consider some set $A_j \in \mathcal{A}$ with $j \in \{0, \dots, m-1\}$. The only jobs which can start in the interval $[(\ell \cdot m + j) \cdot p, (\ell \cdot m + j) \cdot p + n]$ are the dummy jobs $D_{A_j}^\ell$ and the jobs J_{A_j,u_i}^ℓ for $u_i \in A_j$. Because there are only $k-1$ dummy jobs $D_{A_j}^\ell$ but k machines, this implies that there exists at least one machine q such that job J_{A_j,u_i}^ℓ is scheduled at machine q for some $u_i \in A_j$. We know that on machine q one job starts in the interval $[(\ell \cdot m + j) \cdot p, (\ell \cdot m + j) \cdot p + n]$ and has starting time $(\ell \cdot m + j) \cdot p + x_\ell^q$. By construction, this job must be J_{A_j,u_i}^ℓ with $i = x_\ell^q$ which implies that $u_i = u_{x_\ell^q} \in X$. We can conclude that X is a hitting set for I. ◀

Now we have all the pieces to prove Theorem 3.

Proof of Theorem 3. Observation 4 shows that the described reduction can be computed in polynomial time and produces an instance of $P \mid r_j, p_j = p \mid \sum U_j$ with k machines. Lemmas 5 and 6 show that the described reduction is correct. Since Hitting Set is known to be NP-hard [21] and W[2]-hard when parameterized by k [10], the result follows. ◀

4 New Analysis of Known Algorithm for $P \mid r_j, p_j = p \mid \sum w_j U_j$

With Theorem 3 we have established that $P \mid r_j, p_j = p \mid \sum U_j$ is NP-hard. Hence, it is natural to resort to parameterized algorithms for efficiently finding exact solutions in restricted cases. To the best of our knowledge, the only known parameterized algorithm for $P \mid r_j, p_j = p \mid \sum w_j U_j$ is an XP-algorithm for the number m of machines as a parameter by Baptiste et al. [2, 3].

▶ **Theorem 7** ([2, 3]). *$P \mid r_j, p_j = p \mid \sum w_j U_j$ can be solved in $n^{O(m)}$ time, where n is the number of jobs and m is the number of machines.*

Theorem 7 implies that $P \mid r_j, p_j = p \mid \sum w_j U_j$ is in XP when parameterized by the number m of machines. Since Theorem 3 also shows W[2]-hardness for $P \mid r_j, p_j = p \mid \sum U_j$ parameterized by the number m of machines, the algorithm behind Theorem 7 presumably cannot be improved to an FPT-algorithm.

However, as it turns out, we can upper-bound the running time of the algorithm behind Theorem 7 in different ways to obtain additional tractability results. In the remainder of this section, we show that the algorithm developed by Baptiste et al. [2, 3] additionally to Theorem 7 also implies the following.

▶ **Theorem 8.** *$P \mid r_j, p_j = p \mid \sum w_j U_j$ can be solved in $p^{O(m)} \cdot n^{O(1)}$ time and in $m^{O(p)} \cdot n^{O(1)}$ time, where n is the number of jobs, m is the number of machines, and p is the processing time.*

Theorem 8 implies that $P \mid r_j, p_j = p \mid \sum w_j U_j$ is in XP when parameterized by the processing time p and that $P \mid r_j, p_j = p \mid \sum w_j U_j$ is in FPT when parameterized by the combination of the number m of machines and the processing time p. In order to prove Theorem 8, we present the dynamic programming algorithm for $P \mid r_j, p_j = p \mid \sum w_j U_j$ by Baptiste et al. [2, 3]. For the correctness of this algorithm, we refer to their work. We give an alternative running time analysis that shows the claimed running time bounds.

To this end, we need to introduce some additional notation and terminology. Recall that $\mathcal{T}$ denotes the set of relevant starting time points. The algorithm makes use of Lemma 2, that is, we can assume the starting times of all jobs in an optimal schedule are from $\mathcal{T}$. A *resource profile* is a vector $x = (x_1, x_2, \dots, x_m)$ with $x_1 \le x_2 \le \dots \le x_m$, $x_m - x_1 \le p$, and $x_i \in \mathcal{T}$ for all $i \in \{1, \dots, m\}$. Let $\mathcal{X}$ denote the set of all resource profiles. Now we define the following dynamic program. We assume that the jobs are sorted according to their due dates, that is, $d_1 \le d_2 \le \dots \le d_n$.

For two resource profiles $a, b \in \mathcal{X}$ and some $k \in \{1, \dots, n\}$ we define $W(k, a, b)$ to be the maximum weighted number of early jobs of any feasible schedule for the jobs $1, \dots, k$ such that

- sorting the starting times of the first jobs on each machine from smallest to largest yields a vector a' with $a \le a'$, and
- sorting the completion times of the last jobs on each machine from smallest to largest yields a vector b' with $b' \le b$,

where for two vectors a, b of length m we say that $a \le b$ if and only if for all $i \in \{1, \dots, m\}$ we have that $a_i \le b_i$.

From this definition, it follows that $W(n, (0, \dots, 0), (t_{\max}, \dots, t_{\max}))$, where $t_{\max}$ is the largest element in $\mathcal{T}$, is the maximum weighted number of early jobs of any feasible schedule. Baptiste et al. [2, 3] proved the following.

▶ **Lemma 9** ([2, 3]). *For all $k \in \{1, \dots, n\}$ and all resource profiles $a, b \in \mathcal{X}$ with $a \le b$ it holds that $W(k, a, b)$ equals $W(k-1, a, b)$ if $r_k \notin [a_1, b_m - p)$ and otherwise*

$$\max\left(W(k-1, a, b), \max_{\substack{x \in \mathcal{X}, r_k \le x_1, x_1 + p \le d_k, a \le x, \\ x' = (x_2, x_3, \dots, x_m, x_1 + p) \le b}} \left(W(k-1, a, x) + W(k-1, x', b) + w_k \right) \right),$$

where we define $W(0, a, b) = 0$ for all $a, b \in \mathcal{X}$ with $a \le b$.

A straightforward running time analysis yields the following. We have that $|\mathcal{T}| \in O(n^2)$ and hence $|\mathcal{X}| \in O(n^{2m})$. It follows that the size of the dynamic programming table W is in $O(n^{4m+1})$ and the time to compute one entry is in $O(n^{2m})$. This together with Lemma 9 yields Theorem 7. In the remainder of the section, we give an alternative running time analysis to prove Theorem 8.

Proof of Theorem 8. To prove Theorem 8 we give a different bound for the size of $\mathcal{X}$. Recall that for all resource profiles $x \in \mathcal{X}$ we have that $x_m - x_1 \leq p$. It follows that there are $|\mathcal{T}|$ possibilities for the value of x_1, and then for $2 \leq i \leq m$ we have that $x_1 \leq x_i \leq x_1 + p$. Hence, we get that $|\mathcal{X}| \in O(n^2 \cdot p^{m-1})$. This together with Lemma 9 immediately gives us that $P \mid r_j, p_j = p \mid \sum w_j U_j$ can be solved in $p^{O(m)} \cdot n^{O(1)}$ time.

Furthermore, we have $x_1 \leq x_2 \leq \ldots \leq x_m$. Hence, the resource profile x can be characterized by counting how many times a value $t \in \mathcal{T}$ appears in x. Again, we can exploit that $x_m - x_1 \leq p$. There are $|\mathcal{T}|$ possibilities for the value of x_1 and given x_1, each other entry x_i of x with $2 \leq i \leq m$ can be characterized by the amount $0 \leq y_i = x_i - x_1 \leq p$ by which it is larger than x_1. Clearly, there are $p+1$ different possible values for the amount y_i. It follows that given x_1, we can characterize x by counting how often a value $0 \leq t \leq p$ appears as an amount. Hence, we get that $|\mathcal{X}| \in O(n^2 \cdot m^{p+1})$. This together with Lemma 9 immediately gives us that $P \mid r_j, p_j = p \mid \sum w_j U_j$ can be solved in $m^{O(p)} \cdot n^{O(1)}$ time. ◀

Lastly, we remark that with similar alternative running time analyses for the dynamic programming algorithm by Baptiste et al. [2, 3], one can show that $P \mid r_j, p_j = p \mid \sum w_j U_j$ is in XP when parameterized by the number of release dates or due dates, and that $P \mid r_j, p_j = p \mid \sum w_j U_j$ is in FPT when parameterized by the combination of the number of machines and the number of release dates or due dates. However, as we show in the next section, we can obtain fixed-parameter tractability by parameterizing only by the number of release dates or parameterizing only by the number of due dates.

5 FPT-Algorithm for $P \mid r_j, p_j = p \mid \sum w_j U_j$

In this section, we present a new FPT algorithm for $P \mid r_j, p_j = p \mid \sum w_j U_j$ parameterized by the number of release dates or due dates. Formally, we show the following.

▶ **Theorem 10.** *$P \mid r_j, p_j = p \mid \sum w_j U_j$ can be solved in $2^{O(2^{r_\#} \cdot r_\#)} \cdot n^{O(1)}$ time and in $2^{O(2^{d_\#} \cdot d_\#)} \cdot n^{O(1)}$ time.*

Theorem 10 implies that $P \mid r_j, p_j = p \mid \sum w_j U_j$ is in FPT when parameterized by the number $r_\#$ of different release dates and when parameterized by the number $d_\#$ of different due dates. We prove the first running time upper-bound of Theorem 10, where we parameterize by the number $r_\#$ of release dates. By Observation 1, the case for the number $d_\#$ of deadlines is symmetric. We present a reduction from $P \mid r_j, p_j = p \mid \sum w_j U_j$ to Mixed Integer Linear Program (MILP).

Mixed Integer Linear Program (MILP)

Input: A vector x of n variables, a subset S of the variables which are considered integer variables, a constraint matrix $A \in \mathbb{R}^{m \times n}$, and two vectors $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$.

Task: Compute an assignment to the variables (if one exists) such that all integer variables in S are set to integer values, $Ax \leq b$, $x \geq 0$, and $c^\intercal x$ is maximized.

We give a reduction that produces an MILP instance with a small number of integer values. More precisely, the number of integer values will be upper-bounded by a function of the number of release dates of the $P \mid r_j, p_j = p \mid \sum w_j U_j$ instance. This allows us to upper-bound the running time necessary to solve the MILP instance using the following well-known result.

▶ **Theorem 11** ([8, 29]). *MILP can be solved in $2^{O(n_{int} \log n_{int})} \cdot |I|^{O(1)}$ time, where n_{int} is the number of integer variables and $|I|$ is the instance size.*

Furthermore, we construct the MILP instance in a way that ensures that there always exist optimal solutions where *all* variables are set to integer values. Informally, we ensure that the constraint matrix for the rational variables is totally unimodular[1]. This allows us to use the following result.

▶ **Lemma 12** ([6]). *Let $A_{frac} \in \mathbb{R}^{m \times n_2}$ be totally unimodular. Then for any $A_{int} \in \mathbb{R}^{m \times n_1}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^{n_1+n_2}$, the MILP*

$$\max c^\intercal x \text{ subject to } (A_{int}\ A_{frac})x \leq b, x \geq 0,$$

where $x = (x_{int}\ x_{frac})^\intercal$ with the first n_1 variables (i.e., x_{int}) being the integer variables, has an optimal solution where all variables are integer.

Before we describe how to construct an MILP instance for a given instance of $P \mid r_j, p_j = p \mid \sum w_j U_j$, we make an important observation on optimal schedules. Intuitively, we show that we can assume that each job is scheduled as early as possible and idle times only happen directly before release dates.

▶ **Lemma 13.** *Let σ be a feasible schedule for an instance of $P \mid r_j, p_j = p \mid \sum w_j U_j$ such that the weighted number of early jobs is W. Then there exists a feasible schedule σ' such that*

- *the weighted number of early jobs is W,*
- *for each job j with $\sigma'(j) = (i,t)$ for some $t \neq r_j$, machine i is not idle at time $t-1$, and*
- *all starting times of σ' are in the set $\mathcal{T}$ of relevant starting time points.*

Proof. Assume that there is a job j such that $\sigma(j) = (i,t)$ for some $t > r_j$ and machine i is idle at time $t-1$. Assume that job j is the earliest such job, that is, the job with minimum t. Since machine i is idle at time $t-1$ and $r_j < t$, we can create a new schedule σ' that is the same as σ except that $\sigma'(j) = (i, t-1)$. Clearly, we have that σ' is feasible and has the same weighted number of early jobs as σ. By repeating this process, we obtain a feasible schedule σ'' with the same set of early jobs and such that for each job j with $\sigma''(j) = (i,t)$ and machine i is idle at time $t-1$, it holds that $t = r_j$. Furthermore, we have that each starting point in σ'' is a release date r or a time t with $t = r + \ell \cdot p$ for some release date r and some integer ℓ. Hence, all starting times of σ'' are in the set $\mathcal{T}$ of relevant starting time points. ◀

We call a feasible schedule σ *release date aligned* if the second condition of Lemma 13 holds, i.e., for each job j with $\sigma(j) = (i,t)$ for some $t > r_j$, the machine i is not idle at time $t-1$. Note that Lemma 13 is stronger than Lemma 2 and implies that there always exists an optimal feasible schedule that is release date aligned.

Given a feasible schedule σ that is release date aligned, we say that a release date r_j is *active* on machine i if job j is scheduled to start at this release date, that is, $\sigma(j) = (i, r_j)$, and machine i is idle at time $r_j - 1$. Let T be a subset of all release dates, then we say that machine i has type T in σ if T is the set of active release dates on machine i.

Recall that $\mathcal{T} = \{t \mid \exists\, r_j \text{ and } \exists\, 0 \leq \ell \leq n \text{ s.t. } t = r_j + p \cdot \ell\}$ denotes the set of relevant starting time points. We say that a starting time $t \in \mathcal{T}$ is available on a machine with type T if $t = r + \ell \cdot p$ for some $r \in T$ and $t + p \leq r'$, where r' is the smallest release date in T that is larger than r.

[1] A matrix is *totally unimodular* if each of its square submatrices has determinant 0, 1, or −1 [9].

Given an instance I of $P \mid r_j, p_j = p \mid \sum w_j U_j$, we create an instance I' of MILP as follows. For each type T we create an integer variable x_T that quantifies how many machines have type T and create the constraint

$$\sum_T x_T \leq m. \tag{1}$$

For each starting time $t \in \mathcal{T}$ we create a fractional variable x_t that quantifies on how many machines the starting time t is available and create the following set of constraints.

$$\forall\, t \in \mathcal{T}: \; x_t = \sum_T t_T \cdot x_T, \tag{2}$$

where $t_T = 1$ if starting time t is available on a machine with type T and $t_T = 0$ otherwise.

For each combination of a job j and a starting time t, we create a fractional variable $x_{j,t}$ if job j can be scheduled to start at time t without violating j's release date or due date. This variable indicates whether job j is scheduled to start at starting time t and is early. We create the following constraints.

$$\forall\, t \in \mathcal{T}: \; \sum_j x_{j,t} \leq x_t. \tag{3}$$

$$\forall\, j \in \{1, \ldots, n\}: \; \sum_t x_{j,t} \leq 1. \tag{4}$$

Finally, we use the following function as the maximization objective.

$$\sum_{j,t} w_j \cdot x_{j,t} \tag{5}$$

This finishes the construction of the MILP instance I'. We can observe the following.

▶ **Observation 14.** *Given an instance I of $P \mid r_j, p_j = p \mid \sum w_j U_j$, the above described MILP instance I' can be computed in $O(2^{r_\#} \cdot (m+n)^2)$ time and has $2^{r_\#}$ integer variables.*

In the following, we prove the correctness of the reduction to MILP. We start with showing that if the $P \mid r_j, p_j = p \mid \sum w_j U_j$ instance admits a feasible schedule where the weighted number of early jobs is W, then the constructed MILP instance admits a feasible solution that has objective value W.

▶ **Lemma 15.** *If the $P \mid r_j, p_j = p \mid \sum w_j U_j$ instance I admits a feasible schedule where the weighted number of early jobs is W, then the MILP instance I' admits a feasible solution that has objective value W.*

Proof. Assume that we are given a feasible schedule σ for I such that the weighted number of early jobs is W. By Lemma 13 we can assume that σ is release date aligned.

We construct a solution for I' as follows. Consider job j and let $\sigma(j) = (i, t)$ such that $t + p \leq d_j$, that is, job j is early. We know that $t \in \mathcal{T}$. We set $x_{j,t} = 1$ and for all $t' \in \mathcal{T}$ with $t' \neq t$, we set $x_{j,t'} = 0$. Note that this guarantees that Constraints (4) are fulfilled. Furthermore, assuming we can set the remaining variables to values such that the remaining constraints are fulfilled, we have that the objective value of the solution to I' is W.

In the remainder, we show how to set the remaining variables such that all constraints are fulfilled. Initially, we set all variables x_T to zero. Next, we determine the type of each machine. Consider machine i. Then $T := \{r \mid \exists j \text{ s.t. } \sigma(j) = (i, r_j)\}$ is the type of machine i. Then we increase x_T by one. We do this for every machine. Clearly, afterwards Constraint (1) are fulfilled.

Next, we set $x_t := \sum_T t_T \cdot x_T$, where $t_T = 1$ if starting time t is available on a machine with type T and $t_T = 0$ otherwise. Clearly, this fulfills Constraints (2). It remains to show that Constraints (3) are fulfilled. So consider some time $t \in \mathcal{T}$. For each job j starting at time t on some machine i, we increased x_T by one for some T with $t_T = 1$ when processing machine i. Thus, we have $\sum_j x_{j,t} \leq \sum_T t_T \cdot x_T = x_t$. ◀

Before we continue with the other direction of the correctness, we prove that we can apply Lemma 12 to show that the MILP instance I' admits an optimal solution where all variables are set to integer values.

▶ **Lemma 16.** *The MILP instance I' admits an optimal solution where all variables are set to integer values. Such a solution can be computed in $2^{O(2^{r_\#} \cdot r_\#)} \cdot n^{O(1)}$ time.*

Proof. Notice that since the Constraints (2) are equality constraints, we have that in any feasible solution to I', all variables x_t are set to integer values. Hence, I' is equivalent to the MILP I'' arising from I' by declaring x_t to be integer variables for every $t \in \mathcal{T}$ (in addition to the variables x_T), and it suffices to show that I'' has an integer solution.

We show that the constraint matrix for the fractional variables $x_{j,t}$ in I'' (i.e., the variables $x_{j,t}$) is totally unimodular. By Lemma 12 this implies that I'' and therefore also I' admits an optimal solution where all variables are set to integer values.

Note that the Constraints (3) partition the set of fractional variables $x_{j,t}$, that is, each fractional variable is part of exactly one of the Constraints (3). The same holds for the Constraints (4). Furthermore, the coefficients in the constraint matrix for each variable are either 1 (if they are part of a constraint) or 0. Hence, we have that the constraint matrix is a 0-1 matrix with exactly two 1's in every column. Moreover, in each column, one of the two 1's appears in a row corresponding to the Constraints (3) and the other 1 is in a row corresponding to the Constraints (4). This is a sufficient condition for the constraint matrix to be totally unimodular [9].

Finally, we argue that an optimal integer solution for I'' can be computed in the claimed running time upper-bound. We use the algorithm implicitly described by Chaudhary et al. [6] in their proof for Lemma 12. First, we compute an optimal solution for I'. By the arguments at the beginning of this proof, this is also an optimal solution for I''. To transform this optimal solution into an optimal solution where every variable is set to an integer value, we fix all integer variables, resulting in an LP[2] instance I''' whose variables are precisely the fractional variables from I''. Since the constraint matrix of this LP I''' is totally unimodular (as argued above), it is well-known that an optimal solution for I''' where all variables are set to integer values can be computed in polynomial time, see e.g. Korte and Vygen [24, Theorem 4.18]. Combining this integral optimal solution for I''' with the integral variables from I'' then yields an optimal solution for I''. Now, with Theorem 11 and Observation 14, we get the claimed overall running time upper-bound. ◀

Now we proceed with showing that if the constructed MILP instance admits an optimal solution that has objective value W, then the original $P \mid r_j, p_j = p \mid \sum w_j U_j$ instance admits a feasible schedule where the weighted number of early jobs is W.

▶ **Lemma 17.** *If the MILP instance I' admits an optimal solution where all variables are set to integer values and that has objective value W, then the $P \mid r_j, p_j = p \mid \sum w_j U_j$ instance I admits a feasible schedule σ where the weighted number of early jobs is W. The schedule σ can be computed from the optimal solution to I' in polynomial time (in the size of I').*

[2] Linear Program (LP) is the special case of MILP where no variables are considered integer variables (that is, the set S in the definition of MILP is empty).

Proof. Assume we are given an optimal solution for I' where all variables are set to integer values and that has objective value W. We construct a feasible schedule σ as follows.

First, we assign a type to every machine. Iterate through the types T and, initially, set $i = 1$. If $x_T > 0$, then assign type T to machines i to $i + x_T - 1$. Afterwards, increase i by x_T. Since the solution to I' fulfills Constraint (1), we know that this procedure does not assign types to more than m machines.

Now iterate through the relevant starting times $i \in \mathcal{T}$. Let $J_t = \{j \mid x_{j,t} = 1\}$ and let $M_t = \{i \mid \text{starting time } t \text{ is available on machine } i\}$. By Constraints (2) and (3) we know that $|J_t| \leq |M_t| = x_t$. Hence, we can create a feasible schedule by setting $\sigma(j) = (i, t)$ for every job $j \in J_t$, where $i \in M_t$ and for all $j, j' \in J_t$ with $j \neq j'$ we have $\sigma(j) = (i, t)$ and $\sigma(j') = (i', t)$ with $i \neq i'$. In other words, we schedule each job in J_t to a distinct machine $i \in M_t$ with starting time t. The Constraints (4) ensure that we schedule each job at most once. For all jobs j that are not contained in any set J_t with $t \in \mathcal{T}$, we schedule j to an arbitrary machine i to a starting time that is later than r_j and later than the completion time of the last job scheduled on i. Clearly, we can compute σ in time polynomial in $|I'|$. By the definition of available starting times and the fact that we only create variable $x_{j,t}$ if $t \geq r_j$, this schedule is feasible.

It remains to show that the weighted number of early jobs is W. To this end, note that we only create variable $x_{j,t}$ if $r_j \leq t$ and $t + p \leq d_j$. Hence, for each job j with $x_{j,t} = 1$ for some $t \in \mathcal{T}$, we know that this job is early in the constructed schedule σ. It follows that the weighted number of early jobs is $\sum_{j,t} w_j \cdot x_{j,t}$, which equals the maximization objective of I and hence equals W. ◀

Now we have all the pieces to prove Theorem 10.

Proof of Theorem 10. Given an instance I of $P \mid r_j, p_j = p \mid \sum w_j U_j$ we create an MILP instance I' as described above and use Lemma 16 to compute an optimal solution for I where all variables are set to integer values. Lemmas 15 and 17 show that we can correctly compute an optimal schedule for I from the solution to I' in polynomial time (in the size of I'). Observation 14 together with Lemma 16 show that this algorithm has the claimed running time upper-bound. ◀

6 Conclusion and Future Work

In this work, we resolved open questions by Kravchenko and Werner [25], Sgall [36], and Mnich and van Bevern [32] by showing that $P \mid r_j, p_j = p \mid \sum U_j$ is NP-hard and W[2]-hard when parameterized by the number of machines. The established hardness of the problem motivates investigating it from the viewpoint of exact parameterized or approximation algorithms. In this work, we focussed on the former, leaving the latter for future research. We provided a first step in systematically exploring the parameterized complexity of $P \mid r_j, p_j = p \mid \sum w_j U_j$. Our parameterized hardness result shows that the known XP-algorithm for the number of machines as a parameter is optimal from a classification standpoint. Furthermore, we showed that this known algorithm implies that the problem is also contained in XP when parameterized by the processing time, and that it is contained in FPT when parameterized by the combination of the number of machines and the processing time. Finally, we give an FPT-algorithm for $P \mid r_j, p_j = p \mid \sum w_j U_j$ parameterized by the number of release dates (or due dates). We leave several questions open, the most interesting one is the following.

- Is $P \mid r_j, p_j = p \mid \sum w_j U_j$ in FPT or W[1]-hard when parameterized by the processing time p of any job?

Other interesting parameters to consider might be the number of early jobs or the number of tardy jobs. It is easy to see that $P \mid r_j, p_j = p \mid \sum w_j U_j$ is in XP when parameterized by either one of those parameters, by some simple guess-and-check algorithm (recall that we can check in polynomial time whether all jobs can be scheduled early [5, 37, 38]). Hence, it remains open whether the problem is in FPT or W[1]-hard with respect to those parameters.

References

1 Esther M. Arkin and Ellen B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18(1):1–8, 1987. doi:10.1016/0166-218X(87)90037-0.

2 Philippe Baptiste. Scheduling equal-length jobs on identical parallel machines. *Discrete Applied Mathematics*, 103(1-3):21–32, 2000. doi:10.1016/S0166-218X(99)00238-3.

3 Philippe Baptiste, Peter Brucker, Sigrid Knust, and Vadim G Timkovsky. Ten notes on equal-processing-time scheduling: at the frontiers of solvability in polynomial time. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 2(2):111–127, 2004.

4 Karl Bringmann, Nick Fischer, Danny Hermelin, Dvir Shabtay, and Philip Wellnitz. Faster minimization of tardy processing time on a single machine. *Algorithmica*, 84(5):1341–1356, 2022. doi:10.1007/S00453-022-00928-W.

5 Peter Brucker and Svetlana A. Kravchenko. Scheduling jobs with equal processing times and time windows on identical parallel machines. *Journal of Scheduling*, 11(4):229–237, 2008. doi:10.1007/S10951-008-0063-Y.

6 Juhi Chaudhary, Hendrik Molter, and Meirav Zehavi. Parameterized analysis of bribery in challenge the champ tournaments. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2704–2712. ijcai.org, 2024. URL: https://www.ijcai.org/proceedings/2024/299.

7 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

8 Daniel Dadush, Chris Peikert, and Santosh Vempala. Enumerative lattice algorithms in any norm via M-ellipsoid coverings. In *Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 580–589. IEEE, 2011. doi:10.1109/FOCS.2011.31.

9 George Bernard Dantzig. *Linear inequalities and related systems*. Number 38. Princeton University Press, 1956.

10 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

11 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

12 Nick Fischer and Leo Wennmann. Minimizing tardy processing time on a single machine in near-linear time. In *Proceedings of the 51st International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 297 of *LIPIcs*, pages 64:1–64:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ICALP.2024.64.

13 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series.* Springer, 2006. doi:10.1007/3-540-29953-X.

14 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

15 Ronald L. Graham, Eugene L. Lawler, Jan K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of Discrete Mathematics*, volume 5, pages 287–326. Elsevier, 1979.

16 Klaus Heeger and Danny Hermelin. Minimizing the weighted number of tardy jobs is W[1]-hard. In *Proceedings of the 32nd Annual European Symposium on Algorithms (ESA)*, volume 308 of *LIPIcs*, pages 68:1–68:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.68.

17 Klaus Heeger, Danny Hermelin, George B Mertzios, Hendrik Molter, Rolf Niedermeier, and Dvir Shabtay. Equitable scheduling on a single machine. *Journal of Scheduling*, 26(2):209–225, 2023. doi:10.1007/S10951-022-00754-6.

18 Danny Hermelin, Yuval Itzhaki, Hendrik Molter, and Dvir Shabtay. On the parameterized complexity of interval scheduling with eligible machine sets. *Journal of Computer and System Sciences*, page 103533, 2024. doi:10.1016/J.JCSS.2024.103533.

19 Danny Hermelin, Shlomo Karhi, Michael L. Pinedo, and Dvir Shabtay. New algorithms for minimizing the weighted number of tardy jobs on a single machine. *Annals of Operations Research*, 298(1):271–287, 2021. doi:10.1007/S10479-018-2852-9.

20 Danny Hermelin, Hendrik Molter, and Dvir Shabtay. Minimizing the weighted number of tardy jobs via (max,+)-convolutions. *INFORMS Journal on Computing*, 2023.

21 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972. doi:10.1007/978-1-4684-2001-2_9.

22 Matthias Kaul, Matthias Mnich, and Hendrik Molter. Single-machine scheduling to minimize the number of tardy jobs with release dates. In *Proceedings of the 19th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 321 of *LIPIcs*, pages 19:1–19:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.IPEC.2024.19.

23 Kim-Manuel Klein, Adam Polak, and Lars Rohwedder. On minimizing tardy processing time, max-min skewed convolution, and triangular structured ilps. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2947–2960. SIAM, 2023. doi:10.1137/1.9781611977554.CH112.

24 Bernhard Korte and Jens Vygen. *Combinatorial Optimization*. Springer, 6th edition edition, 2018.

25 Svetlana A Kravchenko and Frank Werner. Parallel machine problems with equal processing times: a survey. *Journal of Scheduling*, 14:435–444, 2011. doi:10.1007/S10951-011-0231-3.

26 Sven O Krumke, Clemens Thielen, and Stephan Westphal. Interval scheduling on related machines. *Computers & Operations Research*, 38(12):1836–1844, 2011. doi:10.1016/J.COR.2011.03.001.

27 Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

28 Eugene L. Lawler and James M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16(1):77–84, 1969.

29 Hendrik W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983. doi:10.1287/MOOR.8.4.538.

30 Jan K. Lenstra, A.H.G. Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

31 LL Liu, Chi To Ng, and TC Edwin Cheng. Bicriterion scheduling with equal processing times on a batch processing machine. *Computers & Operations Research*, 36(1):110–118, 2009. doi:10.1016/J.COR.2007.07.007.

32 Matthias Mnich and René van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, 100:254–261, 2018. doi:10.1016/J.COR.2018.07.020.

33 James M. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15:102–109, 1968.

34 Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems, 5th Edition*. Springer, 2016.

35 Baruch Schieber and Pranav Sitaraman. Quick minimization of tardy processing time on a single machine. In *Proceedings of the 18th International Symposium on Algorithms and Data Structures Symposium (WADS)*, volume 14079 of *Lecture Notes in Computer Science*, pages 637–643. Springer, 2023. doi:10.1007/978-3-031-38906-1_42.

36 Jirí Sgall. Open problems in throughput scheduling. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA)*, volume 7501 of *Lecture Notes in Computer Science*, pages 2–11. Springer, 2012. doi:10.1007/978-3-642-33090-2_2.

37 Barbara B. Simons. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM Journal on Computing*, 12(2):294–299, 1983. doi:10.1137/0212018.

38 Barbara B. Simons and Manfred K. Warmuth. A fast algorithm for multiprocessor scheduling of unit-length jobs. *SIAM Journal on Computing*, 18(4):690–710, 1989. doi:10.1137/0218048.

39 Shao Chin Sung and Milan Vlach. Maximizing weighted number of just-in-time jobs on unrelated parallel machines. *Journal of Scheduling*, 8(5):453–460, 2005. doi:10.1007/S10951-005-2863-7.