Single-Machine Scheduling to Minimize the Number of Tardy Jobs with Release Dates

Matthias Kaul[1]
Universität Bonn, Bonn, Germany

Matthias Mnich
Hamburg University of Technology, Institute for Algorithms and Complexity, Hamburg, Germany

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

Abstract

We study the fundamental scheduling problem $1 \mid r_j \mid \sum w_j U_j$: schedule a set of n jobs with weights, processing times, release dates, and due dates on a single machine, such that each job starts after its release date and we maximize the weighted number of jobs that complete execution before their due date. Problem $1 \mid r_j \mid \sum w_j U_j$ generalizes both Knapsack and Partition, and the simplified setting without release dates was studied by Hermelin et al. [Annals of Operations Research, 2021] from a parameterized complexity viewpoint.

Our main contribution is a thorough complexity analysis of $1 \mid r_j \mid \sum w_j U_j$ in terms of four key problem parameters: the number $p_\#$ of processing times, the number $w_\#$ of weights, the number $d_\#$ of due dates, and the number $r_\#$ of release dates of the jobs. $1 \mid r_j \mid \sum w_j U_j$ is known to be weakly para-NP-hard even if $w_\# + d_\# + r_\#$ is constant, and Heeger and Hermelin [ESA, 2024] recently showed (weak) W[1]-hardness parameterized by $p_\#$ or $w_\#$ even if $r_\#$ is constant.

Algorithmically, we show that $1 \mid r_j \mid \sum w_j U_j$ is fixed-parameter tractable parameterized by $p_\#$ combined with any two of the remaining three parameters $w_\#$, $d_\#$, and $r_\#$. We further provide pseudo-polynomial XP-time algorithms for parameter $r_\#$ and $d_\#$. To complement these algorithms, we show that $1 \mid r_j \mid \sum w_j U_j$ is (strongly) W[1]-hard when parameterized by $d_\# + r_\#$ even if $w_\#$ is constant. Our results provide a nearly complete picture of the complexity of $1 \mid r_j \mid \sum w_j U_j$ for $p_\#$, $w_\#$, $d_\#$, and $r_\#$ as parameters, and extend those of Hermelin et al. [Annals of Operations Research, 2021] for the problem $1 \mid\mid \sum w_j U_j$ without release dates.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Scheduling algorithms

Keywords and phrases Scheduling, Release Dates, Fixed-Parameter Tractability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.19

Related Version *Full Version*: https://arxiv.org/abs/2408.12967 [16]

Funding *Matthias Mnich*: Partially supported by DFG project MN 59/4-1.
Hendrik Molter: Supported by the European Union's Horizon Europe research and innovation programme under grant agreement 949707.

Acknowledgements We wish to thank Danny Hermelin and Dvir Shabtay for fruitful discussions that led to some of the results of this work.

[1] Most work while affiliated with Hamburg University of Technology, Institute for Algorithms and Complexity, Hamburg, Germany.

1 Introduction

The problem of scheduling jobs to machines is one of the core application areas of combinatorial optimization [25]. Typically, the task is to allocate jobs to machines in order to minimize a certain objective function while complying with certain constraints. In our setting, the jobs are characterized by several numerical parameters: a *processing time*, a *release date*, a *due date*, and a *weight*. We have access to a single machine that can process one job (non-preemptively) at a time. We consider one of the most fundamental objective functions, namely to minimize the weighted number of tardy jobs, where a job is considered *tardy* if it completes after its due date. In the standard three-field notation for scheduling problems by Graham [8], the problem is called $1 \mid r_j \mid \sum w_j U_j$. We give a formal definition in Section 2.

The interest in $1 \mid r_j \mid \sum w_j U_j$ comes from various sources. It generalizes several fundamental combinatorial problems. In the most simple setting, without weights and release dates, the classic algorithm by Moore [24] computes an optimal schedule in polynomial time. When weights are added, the problem $(1 \mid\mid \sum w_j U_j)$ encapsulates the Knapsack problem, a cornerstone in combinatorial optimization and one of Karp's 21 NP-complete problems [15]. Precisely, when all jobs are released at time zero and all jobs have a common due date, we obtain the Knapsack problem. Karp's NP-hardness proof (from his seminal paper [15]) is the first example of a reduction to a problem involving numbers. The problem $1 \mid\mid \sum w_j U_j$ is one of the most extensively studied problems in scheduling and can be solved in pseudopolynomial time by the classic algorithm by Lawler and Moore [19]. Hermelin et al. [14] showed that this algorithm can be improved in various restricted settings. Better running times have been achieved for the special case where the weights of the jobs equal their processing times [3, 6, 17, 26].

The problem $1 \mid\mid \sum w_j U_j$ (so without release dates) has been studied from the perspective of parameterized complexity by Hermelin et al. [13]. They considered the number $p_\#$ of different processing times, the number $d_\#$ of different due dates, and the number $w_\#$ of different weights as parameters and showed fixed-parameter tractability for $w_\# + p_\#$, $w_\# + d_\#$, and $p_\# + d_\#$ as well as giving an XP-algorithm for the parameters $p_\#$ and $w_\#$. These results are presumably tight, as Heeger and Hermelin [9] recently showed (weak) W[1]-hardness for the parameters $p_\#$ and $w_\#$. The problem has also been studied under fairness aspects [10].

The addition of release dates is naturally motivated in every scenario where not all jobs are initially available. The aim of this paper is to study the parameterized complexity of the problem $(1 \mid r_j \mid \sum w_j U_j)$ in this setting. Here, it encapsulates Partition and becomes weakly NP-hard [20], even if there are only two different release dates and two different due dates and all jobs have the same weight. It has previously been studied for the case of uniform processing times, both on a single machine [7] and for parallel machines [2, 11], as well as for the special case of interval scheduling [1, 12, 18, 27].

Our Contributions. In this paper, we deploy the tools of parameterized complexity to study the computational complexity of $1 \mid r_j \mid \sum w_j U_j$. In the spirit of "parameterizing by the number of numbers" [5], we analyze the complexity picture with respect to (i) the number $p_\#$ of distinct processing times of the jobs, (ii) the number $w_\#$ of distinct weights of the jobs, (iii) the number $d_\#$ of distinct due dates of the jobs, and (iv) the number $r_\#$ of distinct release dates of the jobs. Thereby, we extend and complement the results obtained by Hermelin et al. [13] for the case where all release dates are zero.

In summary, we obtain an almost complete classification into tractable cases (meaning that we find a fixed-parameter algorithm) and intractable cases (meaning that we show the problem to be W[1]-hard) depending on which subset of parameters from $\{p_\#, w_\#, d_\#, r_\#\}$ we consider.

First note that for some parameter combinations, the problem complexity has already been resolved. In particular, $1 \mid r_j \mid \sum w_j U_j$ is known to be weakly NP-hard for $d_\# = 1$ and $r_\# = 1$, as this setting captures the Knapsack problem [15]. Furthermore, $1 \mid r_j \mid \sum w_j U_j$ is known to be weakly NP-hard for $d_\# = 2$, $w_\# = 1$, and $r_\# = 2$ [20]. For parameter $p_\#$ as well as $w_\#$, Heeger and Hermelin [9] showed weak W[1]-hardness even if $r_\# = 1$.

That leaves open the parameterized complexity for several parameter combinations. We extend the known hardness results by showing the following:

- $1 \mid r_j \mid \sum w_j U_j$ is (strongly) W[1]-hard parameterized by $d_\# + r_\#$ even if $w_\# = 1$.

This result is obtained by a straightforward reduction showing that $1 \mid r_j \mid \sum w_j U_j$ generalizes Bin Packing. Our main results are on the algorithmic side, where we show the following:

- $1 \mid r_j \mid \sum w_j U_j$ is fixed-parameter tractable parameterized by $p_\# + d_\# + r_\#$.
- $1 \mid r_j \mid \sum w_j U_j$ is fixed-parameter tractable parameterized by $p_\# + w_\# + d_\#$ or $p_\# + w_\# + r_\#$.
- $1 \mid r_j \mid \sum w_j U_j$ can be solved in pseudo-polynomial time for constant $r_\#$ or constant $d_\#$.

For the first two, we employ reductions to Mixed Integer Linear Programming (MILP), and for the latter, we give a dynamic programming algorithm. Due to space constraints, proofs of results marked with $\star$ are deferred to a full version [16].

With our results, we resolve the parameterized complexity of $1 \mid r_j \mid \sum w_j U_j$ for almost all parameter combinations of $\{p_\#, w_\#, r_\#, d_\#\}$ and hence give the first comprehensive overview thereof. The remaining question is whether $1 \mid r_j \mid \sum w_j U_j$ is polynomial-time solvable for *constant* $p_\#$ or fixed-parameter tractable for $p_\# + w_\#$. It also remains open whether $1 \mid r_j \mid \sum w_j U_j$ is fixed-parameter tractable parameterized by $p_\#$ if all numbers are encoded in unary. A technical report [4] claims (strong) NP-hardness even for $p_\# = 2$ and $w_\# = 1$. If that result holds, then it would also settle the parameterized complexity for the parameter combination $p_\# + w_\#$, and for $p_\#$ when all processing times and weights are encoded in unary. Otherwise, those questions would remain open.

Our results contribute to the growing body of investigating the parameterized complexity of fundamental scheduling problems [23]; for reference, we refer to the open problem collection by Mnich and van Bevern [22].

2 Preliminaries

Scheduling. The problem considered in this work is denoted $1 \mid r_j \mid \sum w_j U_j$ in the standard three-field notation for scheduling problems by Graham [8]. In this problem, we have n jobs and one machine that can process one job at a time. Each job $j \in \{1, \ldots, n\}$ has a *processing time* p_j, a *release date* r_j, a *due date* d_j, and a *weight* w_j, where we p_j, r_j, d_j, and w_j are non-negative integers. We use $p_\#$, $r_\#$, $d_\#$, and $w_\#$ to denote the number of different processing times, release dates, due dates, and weights, respectively.

A *schedule* $\sigma : \{1, \ldots, n\} \to \mathbb{N}$ assigns to each job j a *starting time* $\sigma(j)$ to process it until its *completion time* $\sigma(j) + p_j$, so no other job $j' \neq j$ must start during j's *execution time* $\sigma(j), \ldots, \sigma(j) + p_j - 1$. We call a job j *early* in a schedule σ if $\sigma(j) + p_j \leq d_j$; otherwise we call job j *tardy*. We say that the machine is *idle* at time s in a schedule σ if no job's execution time contains s. The goal is to find a schedule that minimizes the weighted number of tardy jobs or, equivalently, maximizes the weighted number of early jobs

$$W = \sum_{j \mid \sigma(j) = s \wedge s + p_j \leq d_j} w_j \ .$$

We call a schedule that maximizes the weighted number of early jobs *optimal*. Formally, the problem is defined as follows:

$1 \mid r_j \mid \sum w_j U_j$
Input: A number n of jobs, a list of processing times $(p_1, p_2, \ldots, p_n)$, 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 an optimal schedule, that is, a schedule σ that maximizes $W = \sum_{j \mid \sigma(j)=s \wedge s+p_j \leq d_j} w_j$.

Given an instance I of $1 \mid r_j \mid \sum w_j U_j$, we make the following observation:

▶ **Observation 1.** *Let I be an instance of $1 \mid r_j \mid \sum w_j U_j$ 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 $d'_j = d_{\max} - r_j$ for each job j. Then I admits a schedule where the weighted number of early jobs is W if and only if I' admits a schedule where the weighted number of early jobs is W.*

Observation 1 holds, as we can transform a schedule σ for I into a schedule σ' for I' (with the same weighted number of early jobs) by setting $\sigma'(j) = d_{\max} - \sigma(j) - p_j$. Intuitively, this means that we can switch the roles of release dates and due dates to obtain instances with the same objective value.

Mixed Integer Linear Programming. For several of our algorithmic results, we use reductions to Mixed Integer Linear Programming (MILP). This problem is defined as follows:

Mixed Integer Linear Programming (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.

If all variables are integer variables, the problem is simply called Integer Linear Programming (ILP). Due to Lenstra's well-known result for MILP [21], we have that:

▶ **Theorem 2** ([21]). *MILP is fixed-parameter tractable when parameterized by the number of integer variables.*

3 Hardness Results

In this section, we first discuss known hardness results for $1 \mid r_j \mid \sum w_j U_j$, and then present a novel parameterized hardness result. Observe that for $r_j = 0$ and $d_j = d$ (that is, all jobs have the same deadline), the problem $1 \mid r_j \mid \sum w_j U_j$ is equivalent to Knapsack, which is known to be weakly NP-hard [15]. Further, there is a straightforward reduction from Partition to $1 \mid r_j \mid \sum w_j U_j$ that only uses two release dates and two due dates, and uniform weights [20]. Finally, Heeger and Hermelin [9] recently showed that the special case of $1 \mid r_j \mid \sum w_j U_j$ without release dates is weakly W[1]-hard when parameterized by either $p_\#$ or $w_\#$. Hence, (together with Observation 1) we have that:

▶ **Proposition 3** ([9, 15, 20]). *The problem $1 \mid r_j \mid \sum w_j U_j$ is*
- *weakly* NP*-hard even if $d_\# = 1$ and $r_\# = 1$,*
- *weakly* NP*-hard even if $r_\# = d_\# = 2$ and $w_\# = 1$,*
- *and weakly* W[1]*-hard when parameterized by either $p_\#$ or $w_\#$ even if $r_\# = 1$ or if $d_\# = 1$.*

The reduction from Partition to $1 \mid r_j \mid \sum w_j U_j$ by Lenstra et al. [20] can straightforwardly be extended to a reduction from Bin Packing, which yields the following result:

▶ **Theorem 4** (⋆). *The problem* $1 \mid r_j \mid \sum w_j U_j$ *is strongly* NP*-hard, and strongly* W[1]*-hard when parameterized by* $r_\# + d_\#$*, even if* $w_\# = 1$.

4 $1 \mid r_j \mid \sum w_j U_j$ parameterized by $p_\# + r_\# + d_\#$

In this section, we present the following result.

▶ **Theorem 5.** *The problem* $1 \mid r_j \mid \sum w_j U_j$ *is fixed-parameter tractable when parameterized by* $p_\# + r_\# + d_\#$.

To prove Theorem 5, we present a reduction from $1 \mid r_j \mid \sum w_j U_j$ to MILP that creates instances of MILP where the number of integer variables is upper-bounded by a function of $p_\#$, $r_\#$, and $d_\#$. The result then follows from Theorem 2.

Given an instance of $1 \mid r_j \mid \sum w_j U_j$, we say that two jobs j and j' have the same type if they have the same processing time, the same release date, and the same due date, that is, $p_j = p_{j'}$, $r_j = r_{j'}$, and $d_j = d_{j'}$. Let $\mathcal{T}$ denote the set of all types. Note that we have $|\mathcal{T}| \leq p_\# \cdot r_\# \cdot d_\#$. Furthermore, we sort all release dates and due dates such that if the k^{th} release date equals the ℓ^{th} due date, we require the release date to appear later in the ordering. Let $\mathcal{L}$ denote an ordered list of all release dates and due dates that complies to the aforementioned requirement. Note that we have $|\mathcal{L}| \leq r_\# + d_\#$.

We now create an integer variable $x^t_{a,b}$ for all $t \in \mathcal{T}$ and all $a, b \in \mathcal{L}$ with $a < b$. Intuitively, if a and b are consecutive in $\mathcal{L}$, then this variable tells us how many jobs of type t to schedule in the time interval $[a, b]$. If a and b are not consecutive in $\mathcal{L}$, then $x^t_{a,b}$ is a zero-one variable that tells us whether we schedule a job of type t in a way such that its processing time intersects all $c \in \mathcal{L}$ with $a < c < b$.

We create the following constraints. The first set of constraints is

$$\forall t \in \mathcal{T} : \sum_{a,b \in \mathcal{L} \mid a<b} x^t_{a,b} \leq n_t, \tag{1}$$

where n_t denotes the number of jobs of type t in the $1 \mid r_j \mid \sum w_j U_j$ instance. Intuitively, these constraints ensure that we do not try to schedule more jobs of type t than there are available.

The second set of constraints is

$$\forall a, b \in \mathcal{L} \text{ with } a < b \text{ and } \forall t \in \mathcal{T} \text{ with } r_t > b \text{ or } d_t < a : x^t_{a,b} = 0, \tag{2}$$

where r_t denotes the release date of jobs of type t and d_t denotes the due date of jobs of type t. Intuitively, these constraints prevent us from trying to schedule a job of a certain type into an interval that conflicts with the job's release date or due date.

The third set of constraints is

$$\forall c \in \mathcal{L} : \sum_{t \in \mathcal{T}} \sum_{a,b \in \mathcal{L} \mid a<c<b} x^t_{a,b} \leq 1 \ . \tag{3}$$

Intuitively, these constraints ensure that for each $c \in \mathcal{L}$ at most one job is scheduled that intersects c.

The fourth set of constraints is

$$\forall a, b \in \mathcal{L} \text{ with } a < b:$$

$$\sum_{t \in \mathcal{T}} \left(\sum_{a', b' \in \mathcal{L} | a' \geq a, b' \leq b, a' < b'} p_t \cdot x^t_{a',b'} + \sum_{a', b' \in \mathcal{L} | a' < a, b' > b} (b-a) \cdot x^t_{a',b'} \right) \leq b - a, \tag{4}$$

where p_t denotes the processing time of jobs of type t. Intuitively, these constraints make sure that for every interval $[a, b]$ with $a, b \in \mathcal{L}$ we do not schedule jobs with total processing time more than $b - a$ into that interval.

Now we specify the objective function. If we want to schedule a certain number of jobs with type t early, we take the ones with the largest weight in order to minimize the weighted number of tardy jobs. Let $x^t = \sum_{a,b \in \mathcal{L}; a<b} x^t_{a,b}$. Intuitively, x^t is the number of jobs with type t that are scheduled early. For each type, $t \in \mathcal{T}$, order the jobs of type t in the $1 \mid r_j \mid \sum w_j U_j$ instance by their weight (largest to smallest) and let w^t_i denote the i^{th} largest weight of jobs with type t. The objective function we aim to maximize is

$$\sum_{t \in \mathcal{T}} \sum_{i=1}^{x^t} w^t_i \ . \tag{5}$$

Note that constraints (1), (2), (3), and (4) are linear. The objective function (5) is convex [13] and it is known that we can obtain an equivalent MILP with a linear objective function at the cost of introducing additional fractional variables and constraints [13]. Furthermore, we can observe the following:

▶ **Observation 6.** *The number of integer variables in the created MILP instance is in* $O(p_\# \cdot r_\# \cdot d_\# \cdot (r_\# + d_\#)^2)$.

Next, we show the correctness of the reduction.

▶ **Lemma 7** (⋆). *If the* $1 \mid r_j \mid \sum w_j U_j$ *instance admits a schedule where the weighted number of early jobs is* W*, then the created MILP instance admits a feasible solution that has objective value at least* W*.*

▶ **Lemma 8.** *If the created MILP instance is feasible and admits a solution with objective value* W*, then the* $1 \mid r_j \mid \sum w_j U_j$ *instance admits a schedule where the weighted number of early jobs is at least* W*.*

Proof. Suppose we are given a solution to the MILP instance with objective value W. We create a schedule σ as follows.

We iterate through pairs $a, b \in \mathcal{L}$ with $a < b$ as follows. We start with the smallest element $a \in \mathcal{L}$ and the smallest element $b \in \mathcal{L}$ with $a < b$ according to the ordering. We maintain a current starting point s that is initially set to $s = a$. Furthermore, we consider all jobs initially as "unscheduled". We proceed as follows.

- We iterate through all $t \in \mathcal{T}$ in some arbitrary but fixed way. If $x^t_{a,b} > 0$, take the $x^t_{a,b}$ unscheduled jobs of type t with the maximum weight and schedule those between s and $s + x^t_{a,b} \cdot p_t$, where p_t is the processing time of jobs of type t. Now consider those jobs scheduled and set $s \leftarrow s + x^t_{a,b} \cdot p_t$. Continue with the next type.
- Replace b with the next-larger element in $\mathcal{L}$. If b is the largest element in $\mathcal{L}$, then replace a with the next-larger element in $\mathcal{L}$, set b to the smallest element $b \in \mathcal{L}$ with $a < b$ according to the ordering, and set $s \leftarrow \max\{a, s\}$. If a is the largest element of $\mathcal{L}$, terminate the process. Otherwise, go to the first step.

Since the solution to the MILP obeys constraint (1), we have that there are sufficiently many jobs of each type that can be scheduled. All jobs that remained unscheduled after the above-described procedure are scheduled in some arbitrary but feasible way.

We claim that the above procedure produces a schedule where the weighted number of early jobs is at least W. We start by showing that the procedure indeed produces a schedule. Clearly, there are no two jobs in conflict in the produced schedule. Furthermore, since we always have $s \geq a$ and since the solution to the MILP obeys constraints (2), we have that no job is scheduled to start before their release date.

In the remainder of the proof, we show that we schedule $x^t = \sum_{a,b \in \mathcal{L}; a<b} x^t_{a,b}$ jobs of type t early. In particular, we schedule the x^t jobs of type t with the largest weights early. As the solution to the MILP has objective value W, it follows that the total weight of early jobs is at least W.

We show that all jobs scheduled in the first step of the above-described procedure are early. To this end, we identify where the procedure inserts idle times and argue that all jobs scheduled in the first step between two consecutive idle times by the procedure are early. An idle time is inserted by the procedure whenever we set $s \leftarrow \max\{a, s\}$ and we have $a > s$. Consider the case where we set $s \leftarrow \max\{a, s\} = a$ for some $a \in \mathcal{L}$. (Note that, for technical reasons, this includes the case where $a = s$.) Let $a' \in \mathcal{L}$ denote the next larger element of $\mathcal{L}$ for which we set $s \leftarrow \max\{a', s\} = a'$. Now suppose, for the sake of contradiction, that there is a job with a starting time between a and a' that is scheduled in the first step of the procedure and is tardy. Let j be the first such job, that is, the one with the smallest starting time. Let $x^t_{a'',b} \geq 1$ with some $a \leq a'' \leq a'$ be the variable that was considered by the procedure when j was scheduled. Then we have that $d_j \geq b$, since otherwise constraints (2) are not met. We make the following case distinction:

1. If $a = a''$, then the completion time of j is at most $a + \sum_{t \in \mathcal{T}} \sum_{b' \in \mathcal{L} | b' \leq b, a < b'} p_t \cdot x^t_{a,b'}$. However, since constraints (4) are met by the solution to the MILP, in particular the one for $a, b \in \mathcal{L}$, we have that
$$a + \sum_{t \in \mathcal{T}} \sum_{b' \in \mathcal{L} | b' \leq b, a < b'} p_t \cdot x^t_{a,b'} \leq b \ .$$
Since $b \leq d_j$, this contradicts the assumption that j is tardy.
2. Assume that $a < a'' \leq a'$. We argue that in this case, for all $x^{t'}_{a''',b'}$ with $t' \in \mathcal{T}$ and $a''', b' \in \mathcal{L}$ with $a \leq a''' < a''$ and $b' > b$ we must have that $x^{t'}_{a''',b'} = 0$. Assume that $x^{t'}_{a''',b'}$ for some $t' \in \mathcal{T}$ and $a''', b' \in \mathcal{L}$ with $a \leq a''' < a''$ and $b' > b$. Consider the constraint (4) for $a'', b \in \mathcal{L}$. Then we have
$$\sum_{t \in \mathcal{T}} \left(\sum_{a', b' \in \mathcal{L} | a' \geq a'', b' \leq b, a' < b'} p_t \cdot x^t_{a',b'} + \sum_{a', b' \in \mathcal{L} | a' < a'', b' > b} (b - a'') \cdot x^t_{a',b'} \right) \leq b - a'' \ .$$
However, we also have that
$$\begin{aligned} b - a'' &< p_t \cdot x^t_{a'',b} + (b - a'') \cdot x^{t'}_{a''',b'} \\ &\leq \sum_{t \in \mathcal{T}} \left(\sum_{a', b' \in \mathcal{L} | a' \geq a'', b' \leq b, a' < b'} p_t \cdot x^t_{a',b'} + \sum_{a', b' \in \mathcal{L} | a' < a'', b' > b} (b - a'') \cdot x^t_{a',b'} \right), \end{aligned}$$
contradicting the assumption that $x^{t'}_{a''',b'} \geq 1$. It follows that the completion time of job j is at most $a + \sum_{t \in \mathcal{T}} \sum_{a', b' \in \mathcal{L} | a' \geq a, b' \leq b, a' < b'} p_t \cdot x^t_{a',b'}$. Since constraint (4) is met for $a, b \in \mathcal{L}$, we have that
$$a + \sum_{t \in \mathcal{T}} \sum_{a', b' \in \mathcal{L} | a' \geq a, b' \leq b, a' < b'} p_t \cdot x^t_{a',b'} \leq b \ .$$

Since $b \leq d_j$, this contradicts the assumption that j is tardy.
We conclude that all jobs scheduled by the above-described procedure in the first step are early. Since for every $a, b \in \mathcal{L}$ and $t \in \mathcal{T}$, the procedure schedules the $x^t_{a,b}$ jobs of type t with the largest weights early, we have that the total weight of early jobs is at least W. This concludes the proof. ◀

Now we have all the pieces available that are needed to prove Theorem 5.

Proof of Theorem 5. Theorem 5 follows directly from Observation 6, Lemmas 7 and 8, and Theorem 2. ◀

5 $1 \mid r_j \mid \sum w_j U_j$ parameterized by $p_\# + w_\# + d_\#$

In this section, we present the following result.

▶ **Theorem 9.** *The problem $1 \mid r_j \mid \sum w_j U_j$ is fixed-parameter tractable when parameterized by $p_\# + w_\# + r_\#$ or when parameterized by $p_\# + w_\# + d_\#$.*

We show the second part of Theorem 9, that is, $1 \mid r_j \mid \sum w_j U_j$ is fixed-parameter tractable when parameterized by $p_\# + w_\# + d_\#$. By Observation 1, from this immediately follows that $1 \mid r_j \mid \sum w_j U_j$ is also fixed-parameter tractable when parameterized by $p_\# + w_\# + r_\#$. To prove the second part of Theorem 9, present a reduction from $1 \mid r_j \mid \sum w_j U_j$ to MILP. Given an instance of $1 \mid r_j \mid \sum w_j U_j$, we create a number of instances of MILP where in each of them, the number of integer variables is upper-bounded by a function of $p_\#$, $w_\#$, and $d_\#$. We solve each instance using Theorem 2 and prove that the $1 \mid r_j \mid \sum w_j U_j$ instance admits a schedule where the weighted number of early jobs is at least W if and only if one of the generated instances admits a solution with objective value at least W. Furthermore, we can upper-bound the number of created MILP instances by a function of $p_\#$, $w_\#$, and $d_\#$.

Given an instance of $1 \mid r_j \mid \sum w_j U_j$, we say that two jobs j and j' have the same type if they have the same processing time, the same weight, and the same due date, that is, $p_j = p_{j'}$, $w_j = w_{j'}$, and $d_j = d_{j'}$. Let $\mathcal{T}$ denote the set of all types. Note that we have $|\mathcal{T}| \leq p_\# \cdot w_\# \cdot d_\#$. For some $r \in \mathbb{N}_0$ and some $t \in \mathcal{T}$ we denote by $t(r)$ the set of jobs with type t whose release date is at least r. Further, we denote by p_t the processing time of jobs with type t. Let $d_1, \ldots, d_{d_\#}$ be the sorted sequence of due dates. We denote with d_ℓ the ℓ^{th} due date and we use d_j to denote the due date of job j (same with release dates). To keep the notation concise we will use $d_0 = 0$ occasionally.

We fix some optimal schedule $\sigma : \{1, \ldots, n\} \to \mathbb{N}$ for the instance so that we may guess some part of it by enumeration. If for a job j and a due date d_ℓ we have $\sigma(j) < d_\ell < \sigma(j) + p_j$, we will say that the job *overlaps* the due date. Notice that in any schedule, any due date is overlapped by at most one job, but a job may overlap multiple due dates.

We now want to enumerate all possible ways due dates can be overlapped by early jobs from some type $t \in \mathcal{T}$ in σ. That is, we consider all ways to partition $d_1, \ldots, d_{d_\#}$ into subsequences of consecutive due dates. There are $2^{d_\#}$ such partitions. For each such subsequence S we consider all job types that might be scheduled overlapping all due dates in S. For the subsequences containing only a single due date, we also consider the case that no job overlaps that due date. This gives $p_\# \cdot w_\# \cdot d_\# + 1$ choices for each subsequence, of which there are $d_\#$. Thus we end up with at most $2^{d_\#} \cdot d_\#^{p_\# \cdot w_\# \cdot d_\# + 1}$ possible *overlap structures* to consider. By enumerating them it is now possible to assume that we know which overlap structure is present in σ.

We make a small simplification at this stage. Suppose we know that some sequence of due dates $d_a, d_{a+1}, \ldots, d_b$ is overlapped by the same job. Then σ schedules every job with one of these due dates such that it is either scheduled to end before d_a, or it is late. Therefore, we can decrease the due date of such jobs to be d_a without changing the optimality of σ. We may thus assume that every job overlaps at most one due date. So suppose that for each d_ℓ we are given the job type $T(d_\ell) = t$ overlapping that due date, or an assertion that no job overlaps d_ℓ, represented by $T(d_\ell) = \emptyset$. If the due date of jobs of type $T(d_\ell)$ is at most d_ℓ we can reject the arrangement immediately, so assume that the due date of jobs of type $T(d_\ell)$ is larger than d_ℓ. We can formulate a MILP that computes an optimal schedule under the constraint that this structure of overlaps is respected. It uses the following variables:

1. $x_t^\ell \in \mathbb{N}_0$ counts the number of jobs of type t to be scheduled between $d_{\ell-1}$ and d_ℓ.
2. $o_a^\ell, o_b^\ell \in \mathbb{N}_0$ are the portions of the job scheduled overlapping d_ℓ that is processed before and after d_ℓ, respectively.
3. $x_j \in [0,1]$ indicates whether job j is scheduled. These variables are fractional, but we will be able to show that they can be rounded.

We could omit generating variables x_t^ℓ that are known to schedule jobs late, that is, those where jobs of type t have a due date that is earlier than d_ℓ. For simplicity, we keep these variables, but one can assume them to be set to 0.

The MILP needs the following sets of constraints. The first constraint sets handle the overlaps around due dates:

$$\forall \ell \in \{1, \ldots, d_\# - 1\} : o_a^\ell + o_b^\ell = p_t, \text{ if } T(d_\ell) = t. \tag{6}$$

$$\forall \ell \in \{1, \ldots, d_\# - 1\} : o_a^\ell + o_b^\ell = 0, \text{ if } T(d_\ell) = \emptyset. \tag{7}$$

$$\forall \ell \in \{1, \ldots, d_\# - 1\} : d_\ell + o_b^\ell \leq d_{\ell+1}. \tag{8}$$

$$\forall \ell \in \{1, \ldots, d_\# - 1\} : d_{\ell-1} + o_a^\ell \leq d_\ell \ . \tag{9}$$

The next set of constraints ensures that we do not try to schedule more jobs of a certain type than there are available:

$$\forall t \in \mathcal{T} : \sum_{j \text{ has type } t} x_j = \sum_{\ell \in \{1,\ldots,d_\#\}} x_t^\ell + |\{d_\ell \mid T(d_\ell) = t\}| \ . \tag{10}$$

The next two sets of constraints, intuitively, ensure that we respect the release dates of the jobs, and that we do not schedule too many jobs between two consecutive due dates.

$$\forall \ell \in \{1, \ldots, d_\#\}, \ell' \in \{1, \ldots, r_\#\} \text{ with } r_{\ell'} \leq d_\ell$$
$$o_a^\ell + \sum_{t \in \mathcal{T}} p_t \cdot \left(\sum_{j \in t(r_{\ell'})} x_j - \sum_{\ell'' > \ell} x_t^{\ell''} - |\{d_{\ell''} \mid T(d_{\ell''}) = t, \ell'' \geq \ell\}| \right) \leq d_\ell - r_{\ell'} \ . \tag{11}$$

$$\forall \ell \in \{1, \ldots, d_\#\} : o_a^\ell + o_b^{\ell-1} + \sum_{t \in \mathcal{T}} p_t \cdot x_t^\ell \leq d_\ell - d_{\ell-1} \ . \tag{12}$$

The objective function (to be maximized) of the MILP is simply

$$\sum_j w_j \cdot x_j \ . \tag{13}$$

We observe the following:

▶ **Observation 10.** *The number of created MILP instances is in* $O(2^{d_\#} \cdot d_\#^{p_\# \cdot w_\# \cdot d_\# + 1})$ *and the number of integer variables in each instance is in* $O(p_\# \cdot w_\# \cdot d_\#^2)$.

Next, we show the correctness of the reduction.

▶ **Lemma 11** (⋆). *If the* $1 \mid r_j \mid \sum w_j U_j$ *instance admits a schedule where the weighted number of early jobs is* W*, then one of the created MILP instances admits a feasible solution that has objective value at least* W.

It remains to show that we can construct a schedule from a solution to the MILP. We will first show some auxiliary result for the case of a single due date.

▶ **Lemma 12.** *Consider any instance* I *of* $1 \mid r_j \mid \sum w_j U_j$ *with a common due date* d*. If for all release dates* r_j *we have*

$$\sum_{j' \in \mathcal{I} \mid r_{j'} \geq r_j} p_{j'} \leq d - r_j \ .$$

then we can schedule all jobs early.

Proof. The statement holds if the instance I contains a single job. Otherwise, we apply induction on the number of jobs. Let j^* be a job with maximum release date. We schedule that job at time $d - p_{j^*}$. This yields a new instance I' of $1 \mid r_j \mid \sum w_j U_j$ with one fewer job and common due date $d - p_{j^*}$. For any of the release dates r'_j of this new instance, it holds that

$$\sum_{j' \in I' \mid r_{j'} \geq r'_j} p_{j'} = \sum_{j' \in I' \mid r_{j'} \geq r'_j} p_{j'} - p_{j^*} \leq d - r'_j - p_{j^*} \ .$$

The lemma statement follows. ◀

▶ **Lemma 13.** *If one of the created MILP instances admits a solution that has objective value* W*, then the* $1 \mid r_j \mid \sum w_j U_j$ *instance admits a schedule where the weighted number of early jobs is at least* W.

Proof. Assume we are given a feasible solution with objective value W to one of the MILP instances. We will begin by rounding the x_j such that they are integral. Suppose there is some job j with $x_j \in (0, 1)$. Due to constraint (10) there exists some other job j' with the same type as j and with $x_{j'} \in (0, 1)$. Assume, without loss of generality, that j has an earlier release date than j'. Then we claim that setting $x_j := \min\{1, x_j + x_{j'}\}$ and $x_{j'} := \max\{0, x_{j'} + x_j - 1\}$ maintains feasibility. Note that $x_j + x_{j'}$ does not change, $x_{j'}$ decreases, and in any constraint where x_j occurs, $x_{j'}$ occurs also because j has the earlier release date and they have the same type. Therefore, the left-hand sides of constraint (10) are unchanged, and those of constraint (11) do not increase, maintaining feasibility. Since j and j' have the same type, the objective value is unchanged. By repeating this rounding operation, we can increase the number of integral x_j until all of them are integral. The same argument can also be used to ensure that no job j has $x_j = 1$ if there is another job j' with the same type and an earlier release date having $x_{j'} = 0$.

We now show how to schedule every early job j with $x_j = 1$. Note that this implies that the weighted number of early jobs is at least W. First, let $J_t^1, \ldots, J_t^{r_\#}$ be the list of jobs with type t sorted by their release dates, and omitting all jobs with $x_j = 0$. It will suffice to prove that we can schedule the last $x_t^{d_\#}$ jobs from each list into the interval $\mathcal{I} := [d_{d_\#-1} + o_b^{d_\#-1}, d_{d_\#}]$, and that we can then schedule a job of type $T(d_{d_\#-1})$ into $[d_{d_\#-1} - o_a^{d_\#-1}, d_{d_\#-1} + o_b^{d_\#-1}]$.

Notice that constraint (11) simplifies for $d_{d_\#}$ to be $\sum_{t\in\mathcal{T}}\sum_{j\in t(r_{\ell'})} p_t \cdot x_j \leq d_{d_\#} - r_{\ell'}$. Further, for the purposes of scheduling into $\mathcal{I}$, we may consider the $x_t^{d_\#}$ jobs from each type t with the latest release dates to have release date at least $d_{d_\#-1} + o_b^{d_\#-1}$, which transforms constraint (12) into $\sum_{t\in\mathcal{T}}\sum_{j\in t(d_{d_\#-1}+o_b^{d_\#-1})} p_t \cdot x_j \leq d_{d_\#} - d_{d_\#-1} - o_b^{d_\#-1}$. From Lemma 12 we see that we can indeed schedule all the required jobs into $\mathcal{I}$.

Remove all the jobs we just scheduled from the instance, and update the MILP solution by setting to 0 the $x_t^{d_\#}$, as well as all x_j for the scheduled jobs. Now we need to schedule the job j^* with the latest due date from the jobs of type $T(d_{d_\#-1})$ into $[d_{d_\#-1} - o_a^{d_\#-1}, d_{d_\#-1} + o_b^{d_\#-1}]$. By constraints (6)-(9) we see that the interval has the correct length for the job, and that the job will not be late. We only need to ensure that it is not scheduled before its release date r_{j^*}. We use constraint (11) to observe

$$o_a^{d_\#-1} + \sum_{t\in\mathcal{T}} p_t \cdot \left(\sum_{j\in t(r_{j^*})} x_j - \sum_{\ell''>d_\#-1} x_t^{\ell''} - |\{d_{\ell''} \mid T(d_{\ell''}) = t, \ell'' \geq d_\# - 1\}| \right) \leq d_{d_\#-1} - r_{j^*}$$

$$\implies r_{j^*} - p_{j^*} + \sum_{t\in\mathcal{T}} p_t \cdot \sum_{j\in t(r_{j^*})} x_j \leq d_{d_\#-1} - o_a^{d_\#-1}$$

$$\implies r_{j^*} \leq d_{d_\#-1} - o_a^{d_\#-1}.$$

We see that j^* can be scheduled as desired. Now we update the MILP solution again by setting $d_{d_\#-1}$ to $d_{d_\#-1} - o_a^{d_\#-1}$, as well as x_{j^*}, $o_b^{d_\#-1}$, and $o_a^{d_\#-1}$ to 0. We also update $T(d_{d_\#-1}) := \emptyset$. This yields a feasible solution to the MILP of the residual instance where we discard the constraints for the final due date. We can iterate this until all jobs j with $x_j = 1$ have been scheduled. We schedule the remaining jobs in an arbitrary but feasible way. Since the objective value of the solution for the MILP is W, we have that the weighted number of early jobs is also at least W. ◀

Now we have all the pieces available that are needed to prove Theorem 9.

Proof of Theorem 9. Between Lemma 11 and Lemma 13 we see that the MILP is equivalent to the original scheduling problem if the correct overlap structure is chosen. By Observation 10 the number of MILPs we need to solve and the number of integer variables in each MILP depends only on $p_\# + w_\# + d_\#$, and thus Theorem 9 follows from Theorem 2 and Observation 1. ◀

6 Unary $1 \mid r_j \mid \sum w_j U_j$ parameterized by $r_\#$

Recall that $1 \mid r_j \mid \sum w_j U_j$ is *weakly* NP-hard in the case where there is only one due date and one release date. Thus even for instances of $1 \mid r_j \mid \sum w_j U_j$ with constant $r_\#$ or $d_\#$, we cannot expect a polynomial-time algorithm solving them in general. However, we show in the following that such instances can be solved in *pseudo-polynomial time*[2], using dynamic programming, thus generalizing the folklore algorithm for Knapsack. Formally, we prove the following:

[2] A problem can be solved in *pseudo-polynomial* time if it can be solved in polynomial time when all numeric values are encoded unarily.

▶ **Theorem 14.** *The problem* $1 \mid r_j \mid \sum w_j U_j$ *is in* XP *when parameterized by* $r_\#$ *or* $d_\#$ *if all numbers are encoded in unary.*

By Observation 1, we can focus on $r_\#$ as a parameter, and the result for $d_\#$ as a parameter follows. We begin by ordering the release dates and denote with r_ℓ and r_k the ℓ^{th} and k^{th} release date, respectively, and we use r_i and r_j to denote the release date of jobs i and j, respectively (same with due dates). We also use $\leq_d$ to denote a fixed order of the jobs such that their due dates are non-descending. We encode this directly into the indexing of the jobs, that is, $i \leq_d j$ if and only if $i \leq j$.

We can now assume that there exists some optimal schedule such that at every release date r_ℓ there is a job scheduled to start exactly at r_ℓ. This can be ensured by enumerating the possible overlap structures of the optimal schedule with respect to the release dates, as in Section 5. To do this, we need to guess for each release date r_ℓ if there is a job scheduled there and what the completion time of that job is. We know the completion time to be in $[r_\ell + 1, r_\ell + p_{\max}]$, so in reality we will need to solve $(p_{\max})^{r_\#}$ dynamic programs and return the best solution found. This is only a pseudo-polynomial time overhead, so we will suppress it in the following.

Notice that between two consecutive release dates, we can now assume the scheduled early jobs to be ordered according to $\leq_d$, that is, by the earliest due date (all tardy jobs are scheduled later on in a feasible but arbitrary way). If they are not ordered as such two adjacent out-of-order jobs can be swapped while maintaining feasibility.

This structural simplification allows us to write a dynamic program with the following recursive definition of the dynamic programming table:

$$T[j, t_1, \ldots, t_{r_\#}] = \max_{\substack{\ell \text{ with } r_\ell \geq r_j \\ \text{and } r_\ell + t_\ell \leq d_j}} \{T[j-1, t_1, \ldots, t_{r_\#}], T[j-1, t_1, , \ldots, t_\ell - p_j, \ldots, t_{r_\#}] + w_j\}.$$

The base cases for the recursion are:

- $T[0, 0, \ldots, 0] = 0$,
- $T[0, t_1, \ldots, t_{r_\#}] = -\infty$ if any of the t_ℓ are not zero,
- $T[j, \cdot, t_\ell, \cdot] = -\infty$ if $t_\ell > r_{\ell+1} - r_\ell$ or $t_\ell < 0$ for some ℓ.

The entry $T[j, t_1, \ldots, t_{r_\#}]$ intuitively represents the most valuable schedule attainable using only the first j jobs according to $\leq_d$, and with a total scheduled job time of t_ℓ starting at r_ℓ.

We will first notice that the DP-table T has at most $n \cdot (n \cdot p_{\max})^{r_\#}$ finite entries. Furthermore, each of those entries can be computed in time $O(r_\#)$ if we process them in increasing order of the first index. It remains to show that there exists a schedule attaining the value $\max_{t_1, \ldots, t_{r_\#}} (T[|J|, t_1, \ldots, t_{r_\#}])$, as well as that there is some cell $T[|J|, t_1, \ldots, t_{r_\#}]$ that has value at least as high as that of an optimal schedule.

▶ **Lemma 15** (⋆). *If the* $1 \mid r_j \mid \sum w_j U_j$ *instance admits a schedule where the weighted number of early jobs is* W*, then there exists some entry of the dynamic programming table* T *with value at least* W*.*

▶ **Lemma 16** (⋆). *Let* $T[i, t_1, \ldots, t_{r_\#}]$ *be a cell of the dynamic program with finite value* W*. Then there is a schedule where at most the first* i *jobs are early, that attains a weighted number of early jobs* W*, and which schedules a total processing time of* t_ℓ *immediately after release date* r_ℓ*.*

We may now conclude the following:

▶ **Corollary 17.** *Let* $W = \max_{t_1,\ldots,t_{r_\#}} \{T[n, t_1, \ldots, t_{r_\#}]\}$. *Then* W *is the maximum weighted number of jobs that can be scheduled early in the* $1 \mid r_j \mid \sum w_j U_j$ *instance.*

Proof. By Lemma 15 we have that the maximum weighted number of jobs that can be scheduled early in the $1 \mid r_j \mid \sum w_j U_j$ instance is ar least W. However, let $T[n, t_1, \ldots, t_{r_\#}]$ be an entry with value W. Then Lemma 16 guarantees that there exists a schedule for the $1 \mid r_j \mid \sum w_j U_j$ instance such that the weighted number of early jobs is W. ◀

Now we have all the pieces to prove a running time upper bounds for our dynamic-programming algorithm for $1 \mid r_j \mid \sum w_j U_j$.

▶ **Lemma 18.** *The problem* $1 \mid r_j \mid \sum w_j U_j$ *can be solved in time* $O(p_{max}^{r_\#} \cdot n \cdot (n \cdot p_{max})^{r_\#} \cdot r_\#)$.

Proof. We first need to guess the correct overlap structure between jobs and release dates of which there are at most $p_{\max}^{r_\#}$ many. For each of these overlap structures, we then compute the dynamic programming in time $O(n \cdot (n \cdot p_{\max})^{r_\#} \cdot r_\#)$. We finally return the best objective value found in any of the dynamic programming tables. By Corollary 17, this is the maximum weighted number of jobs that can be scheduled early in the $1 \mid r_j \mid \sum w_j U_j$ instance. ◀

Now we have all the pieces to prove Theorem 14.

Proof of Theorem 14. Theorem 14 follows directly from Lemma 18 and Observation 1. ◀

Notice that the proof of Lemma 16 also gives a backtracking procedure which allows us to compute an optimal schedule in time $O(n \cdot r_\#)$ for a given filled in dynamic programming table with the maximum entry already computed.

7 Conclusion

In this work, we give a comprehensive overview of the parameterized complexity of $1 \mid r_j \mid \sum w_j U_j$ for the parameters number $p_\#$ of processing times, number $w_\#$ of weights, number $r_\#$ of release dates, and number $d_\#$ of due dates. We leave several questions for future research, in particular:

- Is $1 \mid r_j \mid \sum w_j U_j$ in XP when parameterized by $p_\#$?
- Is $1 \mid r_j \mid \sum w_j U_j$ fixed-parameter tractable when parameterized by $p_\# + w_\#$?
- Is $1 \mid r_j \mid \sum w_j U_j$ fixed-parameter tractable when parameterized by $p_\#$ and all numbers are encoded unarily?

We remark that a technical report [4] claims (strong) NP-hardness even for $p_\# = 2$ and $w_\# = 1$. If that result holds, then it would also settle the parameterized complexity for the parameter combination $p_\# + w_\#$, and for $p_\#$ even when all processing times and weights are encoded in unary.

References

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

2 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 J. Belgian, French and Ital. Oper. Res. Soc.*, 2(2):111–127, 2004. doi:10.1007/s10288-003-0024-4.

3 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.

4 Jan Elffers and Mathijs de Weerdt. Scheduling with two non-unit task lengths is NP-complete. Technical report, 2014. doi:10.48550/arXiv.1412.3095.

5 Michael R. Fellows, Serge Gaspers, and Frances A. Rosamond. Parameterizing by the number of numbers. *Theory Comput. Syst.*, 50(4):675–693, 2012. doi:10.1007/S00224-011-9367-Y.

6 Nick Fischer and Leo Wennmann. Minimizing tardy processing time on a single machine in near-linear time. Technical report, 2024. doi:10.48550/arXiv.2402.13357.

7 M. R. Garey, D. S. Johnson, B. B. Simons, and R. E. Tarjan. Scheduling unit–time tasks with arbitrary release times and deadlines. *SIAM J. Comput.*, 10(2):256–269, 1981. doi:10.1137/0210018.

8 Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17(2):416–429, 1969. doi:10.1137/0117039.

9 Klaus Heeger and Danny Hermelin. Minimizing the weighted number of tardy jobs is W[1]-hard. In *Proc. ESA 2024*, volume 308 of *Leibniz Int. Proc. Informatics*, pages 68:1–68:14, 2024. doi:10.4230/LIPICS.ESA.2024.68.

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

11 Klaus Heeger and Hendrik Molter. Minimizing the number of tardy jobs with uniform processing times on parallel machines. Technical report, 2024. doi:10.48550/arXiv.2404.14208.

12 Danny Hermelin, Yuval Itzhaki, Hendrik Molter, and Dvir Shabtay. On the parameterized complexity of interval scheduling with eligible machine sets. *J. Comput. Syst. Sci.*, page 103533, 2024. doi:10.1016/J.JCSS.2024.103533.

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

14 Danny Hermelin, Hendrik Molter, and Dvir Shabtay. Minimizing the weighted number of tardy jobs via $(\max, +)$-convolutions. *INFORMS J. Comput.*, 36:836–848, 2024. doi:10.1287/IJOC.2022.0307.

15 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

16 Matthias Kaul, Matthias Mnich, and Hendrik Molter. Single-machine scheduling to minimize the number of tardy jobs with release dates. Technical report, 2024. doi:10.48550/arXiv.2408.12967.

17 Kim-Manuel Klein, Adam Polak, and Lars Rohwedder. On minimizing tardy processing time, max-min skewed convolution, and triangular structured ILPs. In *Proc. SODA 2023*, pages 2947–2960, 2023. doi:10.1137/1.9781611977554.CH112.

18 Sven O. Krumke, Clemens Thielen, and Stephan Westphal. Interval scheduling on related machines. *Comput. Oper. Res.*, 38(12):1836–1844, 2011. doi:10.1016/J.COR.2011.03.001.

19 Eugene L. Lawler and James M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Mgmt. Sci.*, 16(1):77–84, 1969. doi:10.1287/mnsc.16.1.77.

20 Jan K. Lenstra, A.H.G. Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. *Ann. Discrete Math.*, 1:343–362, 1977. doi:10.1016/S0167-5060(08)70743-X.

21 Hendrik W Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/MOOR.8.4.538.

22 Matthias Mnich and René van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Comput. Oper. Res.*, 100:254–261, 2018. doi:10.1016/J.COR.2018.07.020.

23 Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Math. Prog.*, 154(1):533–562, 2015. doi:10.1007/S10107-014-0830-9.

24 James M. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Mgmt. Sci.*, 15:102–109, 1968. doi:10.1287/mnsc.15.1.102.

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

26 Baruch Schieber and Pranav Sitaraman. Quick minimization of tardy processing time on a single machine. In *Proc. WADS 2023*, volume 14079 of *Lecture Notes Comput. Sci.*, pages 637–643, 2023. doi:10.1007/978-3-031-38906-1_42.

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