Journal of Scheduling (2024) 27:565–585
https://doi.org/10.1007/s10951-024-00825-w

Scheduling a single machine with multiple due dates per job

Raphael Kühn[1] · Christian Weiß[1] · Heiner Ackermann[1] · Sandy Heydrich[1]

Accepted: 1 October 2024 / Published online: 23 October 2024

Abstract
In this paper, we consider single-machine scheduling with multiple due dates per job. This is motivated by several industrial applications, where it is not important by how much we miss a due date. Instead the relevant objective is to minimize the number of missed due dates. Typically, this situation emerges whenever fixed delivery appointments are chosen in advance, such as in the production of individualized pharmaceuticals or when customers can only receive goods at certain days in the week, due to constraints in their warehouse operation. We compare this previously unexplored problem with classical due date scheduling, for which it is a generalization. We show that single-machine scheduling with multiple due dates is NP-hard in the strong sense if processing times are job dependent. If processing times are equal for all jobs, then single-machine scheduling with multiple due dates is at least as hard as the long-standing open problem of weighted tardiness with equal processing times and release dates $1 \mid r_j, p_j = p \mid \sum w_j T_j$. Finally, we focus on the case of equal processing times and provide several polynomially solvable special cases as well as an exact branch-and-bound algorithm and heuristics for the general case. Experiments show that our branch-and-bound algorithm compares well to modern exact methods to solve problem $1 \mid r_j, p_j = p \mid \sum w_j T_j$.

Keywords Single-machine scheduling · Multiple due dates · Branch and bound

1 Introduction

In this paper, we consider a new scheduling model, arising from various industrial production planning problems. Applications exist, for example, in the field of personalized medicines or for productions with fixed delivery schedules.

Traditionally, in scheduling models with due dates, it is assumed that if a job misses its due date, either a fixed penalty or a penalty proportionate to the amount of tardiness is incurred. However, in the production of personalized medicines, where every patient is provided with an individual drug, another setup can be observed instead, in particular during clinical trials. Here, patients periodically show up in treatment centers at fixed times to be examined and to receive their drug. According to the study protocol, the drug should be administered during a specific appointment. The date of this appointment (minus a buffer for transportation) is the due date to manufacturing the drug. If a due date cannot be met, then the respective patient receives their drug at their next appointment instead, for example one week later. Thus, if a due date is missed, it can be missed by up to one week without incurring any further negative effect, but if it is missed by more than that, additional penalties may arise.

Similarly, production companies try to make transportation and warehouse operations more robust and plannable by implementing periodic transportation schedules. Additionally, such measures are also used to minimize costs, by ensuring that the company can send out shipments for neighboring regions at the same time. Typically, these shipping schedules are agreed upon with customers, such that a customer can plan their own warehouse operations accordingly. Again, this means that if a due date/shipping date for a job is missed, there may be, e.g., a week until the next expected shipping date. During that week, no additional penalty is incurred by delaying the job further.

✉ Raphael Kühn
raphael.kuehn@itwm.fraunhofer.de

Christian Weiß
christian.weiss@itwm.fraunhofer.de

Heiner Ackermann
heiner.ackermann@itwm.fraunhofer.de

Sandy Heydrich
sandy.heydrich@itwm.fraunhofer.de

[1] Department for Optimization, Fraunhofer Institute for Industrial Mathematics ITWM, 67663 Kaiserslautern, Germany

In order to capture these additional features, we introduce a new scheduling model, where each job can have multiple due dates. A weight is assigned to each due date, representing the penalty that is incurred if the due date is missed. The goal is to minimize the weighted number of missed due dates.

During the production process of our industry partner, the same production steps have to be performed for each customer, while only the input material is different for each production. Since there is a single bottleneck resource, the remaining production steps can be adapted by adapting the release dates and due dates. Therefore, we focus on the single-machine version of the problem in this paper.

Formally, let $\mathcal{J}$ be a set of n jobs $\{J_1, J_2, \ldots, J_n\}$. Each job $J_j \in \mathcal{J}$ has a release date $r_j \in \mathbb{N}$, a processing time $p_j \in \mathbb{N}$ and a set $D_j \subset \mathbb{N}$ of $K_j \in \mathbb{N}$ due dates, $D_j = \{d_{j,1}, d_{j,2}, \ldots, d_{j,K_j}\}$. Additionally, a weight $w_{j,k} \in \mathbb{N}$ is assigned to each due date $d_{j,k} \in D_j$. From now on, we always assume that the due dates of each job are numbered in ascending order, i.e., $d_{j,1} < d_{j,2} < \ldots$ for each job $J_j \in \mathcal{J}$. Furthermore, we assume that the jobs are numbered by ascending release date, i.e., $r_j \leq r_{j+1}$. Also, given a set of jobs $\mathcal{J}$ we define the set of distinct release dates of jobs in $\mathcal{J}$ by $R(\mathcal{J})$. A feasible schedule ς assigns a start time $S_j(\varsigma)$ to each job and a completion time $C_j(\varsigma) = S_j(\varsigma) + p_j$, such that no job is started before its release date and at most one job is processed at any time. The goal is to find a feasible schedule ς, which minimizes the weighted number of missed due dates. To be precise, given the function

$$U_{j,k}(\varsigma) = \begin{cases} 1 & C_j(\varsigma) > d_{j,k} \\ 0 & \text{otherwise,} \end{cases}$$

the objective function is given by

$$W(\varsigma) = \sum_{j=1}^{n} \sum_{k=1}^{K_j} w_{j,k} U_{j,k}(\varsigma). \tag{1}$$

For sake of conciseness, we will not refer to the schedule ς unless necessary and use the shorthands S_j and C_j. In the following, we use the standard notation by Graham et al. (1979) and write $\sum w_{j,k} U_{j,k}$ for the objective function. Then, in the famous three-field notation, the problem of scheduling a single machine to minimize the weighted number of missed due dates can be written as

$$1 \mid r_j \mid \sum w_{j,k} U_{j,k}. \tag{2}$$

Large parts of this paper are dedicated to the special case where all processing times are equal, which is written as

$$1 \mid r_j, p_j = p \mid \sum w_{j,k} U_{j,k}. \tag{3}$$

For convenience, we also introduce the shorthand MDS (multiple due date scheduling problem) to denote problem (2), i.e., scheduling a single machine to minimize the weighted number of missed due dates. The special case (3) with equal processing times is denoted by MDS − EP or, if all processing times are unitary, MDS − UP.

1.1 Our results

Since MDS is a new problem formulation, we first start with a complexity analysis and show that MDS is in general NP-hard. For the special case of MDS − EP, we do not solve the complexity status, but show that it is at least as hard as the long-standing open problem of scheduling problems with the weighted tardiness objective function.

Furthermore, we show that MDS − EP is fixed parameter tractable if the number of different release dates is seen as a parameter.

Finally, we present a branch-and-bound algorithm and heuristics for the equal processing case that we then test on randomly generated problem instances. We extend a famous classical result and show that if job release dates are multiples of the common processing time, it is possible to minimize the weighted number of missed due dates in polynomial time by solving an assignment problem. Using the previous result, we develop a branch-and-bound algorithm, which solves our problem exactly by iteratively solving assignment problems. This branch-and-bound algorithm compares well to some recent state-of-the-art exact algorithms for weighted tardiness scheduling.

1.2 Literature review

To the best of our knowledge, problem MDS has not been studied in the literature before. However, there are several important, related problems, which we consider in this literature review.

Scheduling with due dates has a nearly seventy year long history of research, starting with the result that the earliest-due-date order minimizes the maximum lateness on a single machine (also known as Jackson's rule (Jackson, 1955)). The other positive result is by Moore (1968), who showed that minimizing the number of late jobs on a single machine takes $O(n \log n)$ time (also known as Moore's algorithm). All other traditional scheduling objectives involving due dates, namely the weighted number of late jobs, the total tardiness and the total weighted tardiness, are NP-hard to minimize even on a single machine (Lawler & Moore, 1969; Lawler, 1977; Lenstra et al., 1977).

On the other hand, in the special case where processing times are equal for all jobs, on a single machine the problems of minimizing the weighted number of late jobs and minimizing the total tardiness are solvable in polynomial time, even if

not all jobs are released at the same time, i.e., in the presence of nonzero release dates (Baptiste, 1999, 2000). The complexity status of minimizing the total weighted tardiness on a single machine, when all jobs have equal processing times and arbitrary release dates, remains undecided to this day (Gafarov et al., 2020; van den Akker et al., 2010). Finally, in the special case where all jobs have unitary processing times even the total weighted tardiness problem with release dates can be minimized in polynomial time via an assignment problem (Brucker, 2007). Several of our results for problem MDS − EP presented in Sects. 3, 4 and 5 are based on a generalization of this assignment approach. Unfortunately, apart from the assignment solution, many of the more complicated techniques used to solve traditional due date based problems are often not applicable to problems MDS and MDS − EP, since in the latter, jobs cannot be ordered by their due date.

There are some other strings of research that involve multiple due dates per job. In one, there is both a deadline that must not be violated and a due date which should be met in a good solution (cf. Lawler (1983); Hariri and Potts (1994)). Another one are scheduling problems with assignable due dates where there are multiple possible due dates per job. Which due date is relevant must follow some rule but can be chosen during the optimization. For details, see, e.g., Cheng and Gupta (1989); Gordon and Kubiak (1998); Gordon et al. (2002); Shabtay (2016); Shabtay and Steiner (2006). While MDS can be seen as a generalization of the problems with due dates and deadlines, the freedom to decide which due date is relevant is a fundamental difference compared to MDS.

The production of personalized medicine has received more attention recently, see, for example, Hertrich et al. (2020, 2022). There, the main focus lies on batching constraints that can typically be found when producing personalized medicines. Here, however, we focus on the influence of the appointment schedule.

1.3 Paper overview

The remainder of this paper is structured as follows. In Sect. 2, we show some general results that are used in the subsequent sections and look at the complexity status of MDS and its special cases. In Sect. 3, we show that subproblems of MDS − EP can be solved in polynomial time by solving an assignment problem. In Sect. 4, we present algorithms that can be used to compute exact solutions for the equal processing time case and present a fixed parameter tractability result. In Sect. 5, we turn to scheduling heuristics and propose three different, simple heuristics to solve our problem and compare them in terms of theoretical run times and, if possible, theoretical optimality gaps. These heuristics are compared to the exact solutions in Sect. 6 via computational experiments on both randomly generated instances and instances generated according to instance descriptions used for experiments in other state-of-the-art papers. We extend our results to the weighted tardiness case in Sect. 7. In Sect. 8, we finish this work with a summary and an outlook for possible future research paths.

2 General results

In this section, we present several general results for problem MDS. These are useful for later parts of the paper. First, we consider certain properties of optimal solutions to problem MDS. Then, we show that problem MDS is NP-hard in the strong sense.

2.1 Properties of optimal solutions to problem MDS

First, note that by definition all weights of due dates are nonnegative. This means the objective function $W(\varsigma)$ defined by (1) is regular, i.e., non-decreasing in each completion time. Thus, a given schedule ς can never be improved by adding idle time into it unless a job with later release date can be started earlier, and we obtain the following lemma.

Lemma 1 *There exists an optimal left-shifted schedule ς^* for problem* MDS*, where each job $J_j \in \mathcal{J}$ starts either at its own release date r_j, or at the completion time $C_i(\varsigma^*)$ of some other job $J_i \in \mathcal{J}$.*

Proof We omit the proof for this lemma as it is straightforward. □

Lemma 1 ensures that an optimal schedule among all left-shifted schedules is also optimal among all feasible schedules. Furthermore, in case a real-world application motivates the use of countably many due dates, only finitely many need to be considered by any algorithm since there is always an optimal schedule that finishes before $\max R(\mathcal{J}) + \sum p$.

For problem MDS − EP, Lemma 1 implies that at most n^2 many possible job completion times need to be considered for an optimal solution, if $n \in \mathbb{N}$ is the number of jobs.

Corollary 1 *For problem* MDS − EP*, there exists an optimal solution ς^*, in which all jobs start at some time $S_j \in \mathcal{S}$ and finish at some time $C_j \in \mathcal{C}$, where*

$$\mathcal{S} = \left\{ r_j + \lambda p \mid j, \lambda = 0, \ldots, n-1 \right\}$$

and

$$\mathcal{C} = \left\{ r_j + \lambda p \mid j, \lambda = 1, \ldots, n \right\}.$$

We will later use these sets in the branch-and-bound algorithm to decide at which times a job must start.

2.2 NP-hardness of problem MDS

We now turn to the complexity status of problem MDS. While MDS is defined for problems with release dates, we shortly regard the case $1 \mid\mid \sum w_{j,k}U_{j,k}$, since multiple due dates per job can be applicable in many other scenarios as well. Note first that problem $1 \mid\mid \sum w_{j,k}U_{j,k}$ is a generalization of the traditional scheduling problem $1 \mid\mid \sum w_j U_j$ (scheduling a single machine to minimize the weighted number of late jobs), where each job has only one due date. Thus, and since problem $1 \mid\mid \sum w_j U_j$ is weakly NP-hard (Lawler & Moore, 1969), we immediately get that problem MDS is weakly NP-hard as well. However, it turns out that problem $1 \mid\mid \sum w_{j,k}U_{j,k}$ is in fact NP-hard in the strong sense. This can be shown via a pseudo-polynomial reduction from the strongly NP-hard, traditional scheduling problem $1 \mid\mid \sum w_j T_j$.

Theorem 1 *Problem* MDS *is NP-hard in the strong sense, even if*

- *there are no release dates,*
- *due dates are periodic with a job-independent period, i.e., there is a job-independent time $t \in \mathbb{N}$ such that the due dates of all jobs can be formulated as $d_{j,k} = d_{j,1} + (k-1)t$,*
- *and weights are only dependent on the job, not on the specific due dates, i.e., $w_{j,k} = w_j$ for all $J_j \in \mathcal{J}$ all $d_{j,k} \in D_j$ and some $w_j \in \mathbb{N}$.*

Proof The proof is presented in the appendix. □

Note that in case of equal processing times we can use a polynomial transformation, since at most n^2 completion times need to be considered for each job due to Corollary 1. Therefore, we can define one due date for each of these completion times such that at most n^3 due dates are required for a given problem instance.

Corollary 2 *Problem* $1 \mid r_j, p_j = p \mid \sum w_j T_j$ *polynomially reduces to problem* MDS − EP.

The complexity status of problem $1 \mid r_j, p_j = p \mid \sum w_j T_j$ is still open (Gafarov et al., 2020), so the reduction of Corollary 2 cannot be used to decide the complexity status of MDS − EP. However, problem $1 \mid r_j, p_j = p \mid \sum w_j T_j$ is a long-standing open problem, so the polynomial reduction indicates that finding a polynomial algorithm for MDS − EP is not an easy task. In a recent work, the authors have analyzed problem $1 \mid r_j, p_j = p \mid \sum w_j T_j$ and conjectured that problem $1 \mid r_j, p_j = p \mid \sum w_j T_j$ is NP-hard, but it was hard to proof using standard techniques(Gafarov et al., 2020). The difficulties they found in proving NP-hardness for problem $1|r_j, p_j = p \mid \sum w_j T_j$ similarly carry over, at least in part, to MDS − EP. Thus, the exact complexity status of MDS − EP is left open in this paper and we instead focus on polynomially solvable special cases as well as practical solution algorithms for the general case of MDS − EP (which then of course can also be used to solve problem $1 \mid r_j, p_j = p \mid \sum w_j T_j$).

3 Solving an assignment problem for MDS − EP

In the following, we often use an assignment problem to solve special cases or subproblems of MDS − EP in polynomial time. Since reformulating a scheduling problem as an assignment of jobs to completion times is a well-known technique, we do not include the proof here.

Theorem 2 *Let* Π *be an instance of* MDS − EP *with n jobs. Let* $\bar{\mathcal{C}} \subset \mathcal{C}$ *be a set of n possible, distinct completion times. Then a best possible schedule using only completion times in* $\bar{\mathcal{C}}$ *can be found in* $O(n^3)$ *time (or it can be decided that no feasible schedule with these completion times exists), by solving an assignment problem.*

By Theorem 2, solving problem MDS − EP reduces to selecting the n out of n^2 completion times from set $\mathcal{C}$ which lead to an optimal solution and then solving an assignment problem in $O(n^3)$ time. Indeed, most algorithms in the later sections of this paper, both exact and heuristics, are constructed by choosing a suitable set $R_{\text{rel}} \subset R(\mathcal{J})$ of release dates at which jobs should start, then defining a corresponding set of n job completion times $\bar{\mathcal{C}} \subset \mathcal{C}$ and finally solving an assignment problem by Theorem 2.

The remainder of this subsection is dedicated to constructing a set of n completion times corresponding to a subset $R_{\text{rel}} \subseteq R(\mathcal{J})$ of exactly those release dates at which jobs start. Let Π be an instance of problem MDS − EP with job set $\mathcal{J}$ and let $R_{\text{rel}} \subseteq R(\mathcal{J})$. In what follows, a schedule ς for instance Π is called R_{rel}-*schedule*, if in ς a job is started at each release date $t \in R_{\text{rel}}$, no job is started at any release date $t \in R(\mathcal{J}) \setminus R_{\text{rel}}$, and each job that is not started at some time $t \in R_{\text{rel}}$ is started at the completion time of another job. Note that by definition, any left-shifted schedule is also an R_{rel}-schedule for some set $R_{\text{rel}} \subseteq R(\mathcal{J})$, but not every R_{rel} schedule is necessarily left-shifted, since in R_{rel}-schedules jobs may also start at the release date of other jobs. Furthermore, define by $J(t)$ the number of jobs $J_j \in \mathcal{J}$ with release date no larger than t, i.e., $J(t) = |\{J_j \in \mathcal{J} \mid r_j \leq t\}|$. Finally, for a finite set $N \subset \mathbb{N}$ define the *Next* operator $\text{Next}(t, N) = \min\{n \in N \mid n \geq t\}$. Note that $\text{Next}(t, N) = \infty$ if $t > \max\{n \in N\}$. Then, for $i = 1, 2, \ldots, n-1$ define n possible job completion times

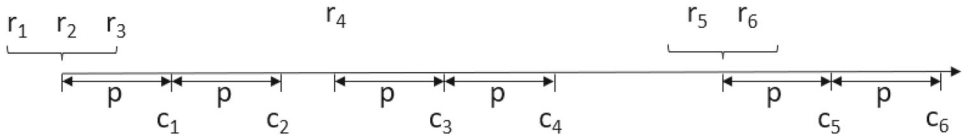


Fig. 1 Example completion times

as follows:

$$c_1 = \min\{r \in R_{\text{rel}}\} + p$$
$$c_{i+1} = \begin{cases} c_i + p, & \text{if } c_i \notin R(\mathcal{J}) \setminus R_{\text{rel}} \\ & \text{and } J(c_i) \geq i+1 \\ & \text{and } c_i + p \\ & \leq \text{Next}(c_i, R_{\text{rel}}) \\ \text{Next}(c_i, R_{\text{rel}}) + p, & \text{otherwise.} \end{cases} \quad (4)$$

Here we define $\infty + p = \infty$. Denote by $\mathcal{C}(R_{\text{rel}})$ the set of completion times defined by Eq. (4).

In Fig. 1, an example for the generation of completion times via Equations (4) is depicted. Let R_{rel} be equal to $\{r_1, r_4, r_5\}$. First the finish time $r_1 + p$ is added. Next, $r_1 + 2p < r_4$ is added. Since $r_1 + 3p > r_4$, the next finish time is added at $r_4 + p$ and not at $r_1 + 3p$. Since only 4 jobs have a release date smaller than $r_4 + 2p$, no finish time is added at $r_4 + 3p$ but at $r_5 + p$.

Theorem 3 *Let* Π *be an instance of problem* MDS $-$ EP *with job set* $\mathcal{J}$.

1. *Let* $R_{\text{rel}} \subseteq R(\mathcal{J})$ *be some set of release dates and let set* $\mathcal{C}(R_{\text{rel}})$ *be defined by Eqn. (4). Then there exists a feasible* R_{rel}*-Schedule, if*
   - $c < \infty$ *for all* $c \in \mathcal{C}(R_{\text{rel}})$*, and*
   - $r + p \in \mathcal{C}(R_{\text{rel}})$ *for each* $r \in R_{\text{rel}}$.
2. *Let* $R_{\text{rel}} \subseteq R(\mathcal{J})$ *be some set of release dates and let* $\mathcal{C}(R_{\text{rel}}) = \{c_1, c_2, \ldots, c_n\}$ *be constructed from* R_{rel} *as above. Then it holds* $J(c_i - p) \geq i$.
3. *Let* $R_{\text{rel}} \subseteq R(\mathcal{J})$ *be some set of release dates. Then* $r_k + p \in \mathcal{C}(R_{\text{rel}})$ *holds for all* $r_k \in R_{\text{rel}}$ *if and only if* $|r_i - r_j| \geq p$ *for all* $r_i, r_j \in R_{\text{rel}}$ *with* $r_i \neq r_j$.
4. *There exists some set of release dates* $R_{\text{rel}} \subseteq R(\mathcal{J})$ *such that there exists an optimal left-shifted* R_{rel}*-schedule with the completion times given by* $\mathcal{C}(R_{\text{rel}})$.

Proof The proof is presented in the appendix. □

4 Equal processing times - Exact solutions

We now turn to problem MDS $-$ EP and some of its special cases, in particular problem MDS $-$ UP. First, we show that problem MDS $-$ UP, or, equivalently, problem MDS $-$ EP with the additional condition that all release dates are multiples of the common processing time p, can be solved in $O(n^3)$ time by modeling it as an assignment problem using the results from Sect. 3. Next, we use the same assignment modeling technique to show that problem MDS $-$ EP is fixed parameter tractable, if the number of different release dates N_r is seen as a parameter.

Finally, we extend the fixed parameter tractability result to versions of problem MDS $-$ EP where the number of different release dates N_r is part of the input, and construct a branch-and-bound algorithm to solve problem MDS $-$ EP in the general case.

4.1 Release dates are multiples of p

In this section, we assume that all release dates are multiples of p, i.e., $r_j = \varrho_j p$ with $\varrho_j \in \mathbb{N}$. This is a sensible assumption, if we consider for example a process where a fixed number of productions are made every day and new input materials arrive each morning. We first show that solving this version of problem MDS $-$ EP reduces to solving problem MDS $-$ UP. Indeed, since for each release date r_j it holds that $r_j = \varrho_j p$ for some $\varrho_j \in \mathbb{N}$, it holds that $\frac{r_j}{p}$ is integer for all jobs. Additionally, due to Lemma 1, it can be assumed that each job starts exactly at its release date or at the finish time of a previous job. Therefore, there exists an optimal schedule were each completion time is a multiple of p. Thus, any due date $d_{j,k}$ that is not a multiple of p can be reduced to the largest multiple of p smaller than $d_{j,k}$, without increasing the objective value of such an optimal solution. Therefore, all time-valued instance parameters can be assumed to be multiples of the common processing time p and we can w.l.o.g. assume that $p = 1$. Thus, problem MDS $-$ EP where all release dates are multiples of the common processing time p reduces to solving problem MDS $-$ UP.

We now show that problem MDS $-$ UP can be solved in $O(n^3)$ time. Note that since $p_j = 1$ for all $J_j \in \mathcal{J}$, there exists an optimal solution where a job is started whenever possible. Indeed, since all release dates are integers and each job has unitary processing time, no job can be started before some release date r_j and finish after it. Thus, at any release date r_j the machine is available to process a job. Furthermore, at each release date r_j at least one job is available to be processed (namely the job J_j that is released at time r_j). Finally, starting a job at time r_j does not prevent us from immediately scheduling a higher-valued job that is released after r_j, since any job started at time r_j is completed before any later jobs are released.

Corollary 3 MDS $-$ UP *can be solved in* $O(n^3)$.

Proof Note that in a schedule with completion times as defined by $\mathcal{C}(R_{\text{rel}})$, a job is started whenever possible in case $R_{\text{rel}} = R(\mathcal{J})$ and $p = 1$. Since there is an optimal schedule such that a job is started whenever possible, we can solve problem MDS − UP by Theorem 2 in $O(n^3)$ time as an assignment problem with the n completion times defined by $\mathcal{C}(R_{\text{rel}})$ and $R_{\text{rel}} = R(\mathcal{J})$. □

4.2 Arbitrary release dates and due dates

In what follows, we discuss exact solution algorithms for the general case of problem MDS − EP. In Sect. 4.2.1, we show that problem MDS − EP is fixed parameter tractable if the number of distinct release dates is seen as a parameter. Then, in Sect. 4.2.2 we provide a branch-and-bound algorithm to solve the general version of problem MDS − EP.

4.2.1 Fixed parameter tractability of Problem MDS — EP

With the results from the previous sections, we can show that problem MDS − EP is fixed parameter tractable when the number of different release dates is viewed as a parameter and not part of the input.

Theorem 4 *The problem* MDS − EP *is fixed parameter tractable when the number of different release dates is seen as a parameter and not part of the input. At most* 2^{N_r} *assignment problems have to be solved to find an optimal solution with* N_r *being the number of different release dates.*

Proof Let Π be an instance of MDS − EP with exactly N_r different release dates, i.e., $|R(J)| = N_r$. This implies that there are 2^{N_r} subsets $R_{\text{rel}} \subset R(\mathcal{J})$. Furthermore, for each subset $R_{\text{rel}} \subset R(J)$ by Theorem 2 and Theorem 3 the best schedule with finish times $\mathcal{C}(R_{\text{rel}})$ can be computed in $O(n^3)$. Also, again by Theorem 3, there is an optimal schedule with finish times defined by $\mathcal{C}(R_{\text{rel}})$ for some $R_{\text{rel}} \subset R(\mathcal{J})$. Thus, to find an optimal schedule for instance Π it is sufficient to compute the best schedule with finish times $\mathcal{C}(R_{\text{rel}})$ for each subset $R_{\text{rel}} \subset R(\mathcal{J})$, which takes at most $O(2^{N_r} \cdot n^3)$ time, which is polynomial, if N_r is fixed. □

4.2.2 A branch-and-bound algorithm for problem MDS — EP

In the following, we construct a branch-and-bound algorithm for solving MDS − EP. Recall that in a R_{rel}-schedule at all times $t \in R_{\text{rel}}$ a job must start and no job is allowed to start at $t \in R(\mathcal{J})\backslash R_{\text{rel}}$. During the branch-and-bound algorithm, multiple relaxed problems are solved. In each of these problems, there may be times in $R(\mathcal{J})$ such that it is not defined if these times belong to R_{rel}, i.e., a job may or may not start at these times. Therefore, we introduce the concept of $(R_{\text{a}}, R_{\text{i}})$-schedules with R_{a} being the set of times at which a job must start and R_{i} being the set of times at which no job is allowed to start. Note that in the following $R_{\text{a}} \subset \mathcal{S}$ and $R_{\text{i}} \subset \mathcal{S}$ with $\mathcal{S}$ as defined in Corollary 1 are not necessarily subsets of $R(\mathcal{J})$ and we assume that the two sets R_{a} and R_{i} are disjoint. Furthermore, we assume that $R(\mathcal{J}) \setminus R_{\text{i}} \neq \emptyset$ since there is always an optimal schedule such that at least one job starts at a release time. We could restrict the sets R_{a} and R_{i} to be subsets of $R(\mathcal{J})$. However, relaxing this condition provides an opportunity for a faster branch-and-bound algorithm. As mentioned before, there may be times $t \in \mathcal{S}$ and especially $t \in R(\mathcal{J})$ such that t is neither in R_{a} nor in R_{i}, and therefore, it is not yet defined if a job has to start at t. We call a schedule a $(R_{\text{a}}, R_{\text{i}})$-schedule if it is feasible, if for each time $r_{\text{act}} \in R_{\text{a}}$ there is a job J_j with $S_j = r_{\text{act}}$ and for each time $r_{\text{ina}} \in R_{\text{i}}$ there is no job J_j with $S_j = r_{\text{ina}}$. Furthermore, in a $(R_{\text{a}}, R_{\text{i}})$-schedule all jobs start at a time $s \in \mathcal{S}$ that is either a release date or the completion time of another job. Since there are only n jobs and for all times $t \in R_{\text{a}}$ a job has to start at t, we assume in the following $|R_{\text{a}}| \leq n$.

We first show how to compute a lower bound for the objective value of the optimal schedule among all $(R_{\text{a}}, R_{\text{i}})$-schedules for two given sets R_{a} and R_{i}. Afterward, we show a branching technique that can be used to construct R_{a} and R_{i} step by step.

In order to show that a lower bound for the optimal schedule among all $(R_{\text{a}}, R_{\text{i}})$-schedules can be computed by an assignment problem, we first derive a lower bound for the start time of each job. To compute these bounds, we need some technical results first. The lower bounds for the start times will be used to adapt the original problem such that the optimal value can be computed with an assignment problem and such that the optimal value will not increase. In the process, we restrict the set of feasible solutions and adapt the release dates of the jobs.

For a given schedule ς and a job J_j, we define $t_{\text{act}}^{\max}(\varsigma, J_j)$ to be the maximum time in $R_{\text{a}} \cup R(\mathcal{J})$ before or equal to S_j such that a job starts at $t_{\text{act}}^{\max}(\varsigma, J_j)$. Similar to the *Next* operator as defined in Sect. 3, we define the *Previous* operator $\text{Prev}(t, N) = \max\{n \in N \mid n \leq t\}$ with $\text{Prev}(t, N) = -\infty$ if $t < \min\{n \in N\}$.

Lemma 2 *Let* Π *be an instance of problem* MDS − EP *with job set* $\mathcal{J}$*, and let* R_{a} *and* R_{i} *be two disjoint sets. For any feasible* $(R_{\text{a}}, R_{\text{i}})$*-schedule* ς *and for all jobs* $J_j \in \mathcal{J}$*, it holds*

1. $t_{act}^{\max}(\varsigma, J_j) \in (R(\mathcal{J}) \cup R_{\text{a}})\backslash R_{\text{i}}$,
2. $t_{act}^{\max}(\varsigma, J_j) \geq \text{Prev}(r_j, R_{\text{a}})$,
3. $\nexists t_i \in R_{\text{i}} : t_i \geq t_{act}^{\max}(\varsigma, J_j), t_i \leq S_j, (t_i - t_{act}^{\max}(\varsigma, J_j)) \mod p = 0$.

Proof The proof can be found in the appendix. □

Given an instance of the MDS − EP and two disjoint sets R_a and R_i, we define the set $T_\mathrm{act}^\mathrm{max}(J_i)$ as follows:

$$\begin{aligned} T_\mathrm{act}^\mathrm{max}(J_i) = \{t \in (R(\mathcal{J}) \cup R_\mathrm{a}) \setminus R_\mathrm{i} | t \\ \geq \mathrm{Prev}(r_j, R_\mathrm{a}), \\ \nexists t_i \in R_\mathrm{i} : t_i \geq t, \\ t_i \leq r_j + (t - r_j) \mod p, \\ (t_i - t) \mod p = 0\}. \end{aligned}$$

Note that $t_\mathrm{act}^\mathrm{max}(\varsigma, J_j)$ defines a specific time for a given schedule and $T_\mathrm{act}^\mathrm{max}(J_i)$ defines a set of times that is independent from any schedule.

Corollary 4 *Let ς be a feasible $(R_\mathrm{a}, R_\mathrm{i})$-schedule. Then it must hold $t_{act}^\mathrm{max}(\varsigma, J_j) \in T_{act}^\mathrm{max}$.*

Proof Let ς be a feasible schedule. Then, with Lemma 2 we know that $t_\mathrm{act}^\mathrm{max}(\varsigma, J_j)$ fulfills all conditions of $T_\mathrm{act}^\mathrm{max}(J_i)$ since the difference between the start time of J_j and $t_\mathrm{act}^\mathrm{max}(\varsigma, J_j)$ must be a multiple of p. □

The set $T_\mathrm{act}^\mathrm{max}(J_i)$ can be used to compute a lower bound for the start time of job J_j. At the start of the branch-and-bound algorithm, this lower bound for the start time is simply r_j. However, by adding times to R_a and R_i one can sometimes conclude that the job J_j cannot start at r_j but at a later time.

Lemma 3 *Let Π be an instance of problem* MDS − EP *with job set $\mathcal{J}$ and R_a and R_i two disjoint sets. We define*

$$S_j^\mathrm{min} := \min_{t \in T_{act}^\mathrm{max}(J_j)} (\max(t, r_j + (t - r_j) \mod p)).$$

For any feasible $(R_\mathrm{a}, R_\mathrm{i})$-schedule ς, it holds

$$S_j^\mathrm{min} \leq S_j(\varsigma).$$

Proof The proof can be found in the appendix. □

Due to the first two conditions of Lemma 2 only $\mathrm{Prev}(r_j, R_\mathrm{a})$ and jobs in $R(\mathcal{J})$ but not in R_a or R_i are candidates for $t_\mathrm{act}^\mathrm{max}(\varsigma, J_j)$. Therefore, the more release dates are assigned to R_a and R_i, the less the candidates for $t_\mathrm{act}^\mathrm{max}(\varsigma, J_j)$ exist with the number of candidates reducing to one if all due dates are assigned to either of the two sets. The previous lemmas provide an opportunity to adapt the release dates of the jobs without increasing the objective value if $S_j^\mathrm{min} > r_j$.
Next, we show how to compute a lower bound for the completion time of the job in the i-th position without knowing the schedule and therefore without knowledge about which job is in the i-th position. These lower bounds will then be used in order to construct a set of possible finish times.
As before, we gather conditions that have to be fulfilled by any job. To find a lower bound, we can take the minimum of all times that fulfill the conditions. Note that not all conditions are necessarily required to compute the lower bound. However, they lead to a higher lower bound and therefore improve the performance of the branch-and-bound algorithm. The first conditions ensure that the distance between the release dates is at least p and that there are always enough jobs to schedule. The next two conditions ensure that a job can be started for all times in R_a. The last condition ensures that the conditions from the previous lemmas can be fulfilled by the computed schedules.

Lemma 4 *Let Π be an instance of problem* MDS − EP *with job set $\mathcal{J}$. Given two disjoint sets R_a and R_i, it holds for the completion times $\mathcal{C}^* = \{c_1^*, c_2^*, ..., c_n^*\}$ of the optimal schedule among all $(R_\mathrm{a}, R_\mathrm{i})$-schedules*

1. $c_i - p \in \mathcal{S}$,
2. $c_1^* \geq \min((R(\mathcal{J}) \cup R_\mathrm{a}) \setminus R_\mathrm{i}) + p$,
3. $c_i^* \geq c_{i-1}^* + p$ *for* $1 < i \leq n$,
4. $\nexists t \in R_\mathrm{a}$ *with* $t < c_i^* < t + p$,
5. $|\{t \in R_\mathrm{a} | t \geq c_i^*\}| \leq n - i$,
6. $J(c_i^* - p) \geq i$,
7. $\exists t_a \in (R(\mathcal{J}) \cup R_\mathrm{a}) \setminus R_\mathrm{i})$ *with*

- $c_i^* \geq t_a + p$,
- $t_a \geq \mathrm{Prev}(c_i^*, R_\mathrm{a})$,
- $(t_a - c_i^*) \mod p = 0$,
- $\nexists t_i \in R_\mathrm{i}$ *with* $t_a \leq t_i \leq c_i^* - p$ *and* $(t_a - t_i) \mod p = 0$.

Proof The proof can be found in the appendix. □

These results can be used in order to compute a lower bound solution. We present an algorithm and show afterwards that it computes a lower bound for the objective value of the optimal $(R_\mathrm{a}, R_\mathrm{i})$-schedule. Note that R_a and R_i are arbitrary sets that are used as an input for the algorithm. We will show later that as long as there is a feasible $R_\mathrm{a}, R_\mathrm{i}$-schedule, the algorithm returns a lower bound for the objective value of the best $R_\mathrm{a}, R_\mathrm{i}$-schedule. The algorithm first computes a set of finish times. Then, it creates a second problem instance that can be solved optimally with the given finish times while every schedule of the original problem remains feasible.
Note that we assume $R(\mathcal{J}) \setminus R_\mathrm{i} \neq \emptyset$ as discussed before.

Algorithm 1 ***Lower bound*** *Given an instance Π of* MDS − EP *and two disjoint sets $R_\mathrm{a}, R_\mathrm{i} \subset \mathcal{S}$.*

1. *If there either exist $t_1, t_2 \in R_\mathrm{a}$ with $|t_1 - t_2| < p$ or if it holds*

   $$J(t) < |\{t' \in R_\mathrm{a} | t' \leq t\}|$$

   for any $t \in R_\mathrm{a}$, set $W_{R_\mathrm{a}, R_\mathrm{i}}^l = \infty$ and terminate the algorithm.

2. *Construct a set of finish times* $\mathcal{C}(R_a, R_i) = \{c_1, \ldots, c_n\}$ *by the following procedure.*

$$c_1 = \begin{cases} \min(R(\mathcal{J}) \setminus R_i) + p & \text{if } R_a = \emptyset \\ \min(R(\mathcal{J}) \setminus R_i) + p & \text{if } \min(R(\mathcal{J}) \setminus R_i) \\ & \quad + p \leq \min(R_a) \\ & \quad \text{and } |R_a| < n, \\ \min(R_a) + p & \text{else.} \end{cases}$$

Construct the remaining completion times as follows. Set $i = 2$.

   (a) *For each* $s \in \mathcal{S}$, *check whether* $c_i = s + p$ *fulfills the conditions 3-6 from Lemma 4 and continue with the next* s *otherwise.*
   (b) *Search for the biggest* $t_{act} \in (R(\mathcal{J}) \cup R_a) \setminus R_i$ *with* $t_{act} \geq \text{Prev}(s, R_a)$, $t_{act} \leq s$ *and* $(t_{act} - s) \mod p = 0$. *If no such* t_{act} *exists go to step (a) and continue with the next* s.
   (c) *Iterate over all* $t_{ina} \in R_i$ *with* $t_{ina} \geq t_{act}$ *and* $t_{ina} \leq s$. *If* $((t_{ina} - t_{act}) \mod p) = 0$ *go to (a) and continue with the next* s.
   (d) *Set* $c_i = s + p$. *If* $i = n$, *stop the algorithm. Otherwise set* $i = i + 1$, *go to (a) and continue with the next* s.

   If less than n *completion times are defined, set* $W^l_{R_a,R_i} = \infty$ *and terminate the algorithm.*
3. *Define a second problem instance* Π' *that is the same as* Π. *Then, adapt the release dates in* Π' *such that*

$$r'_j = \begin{cases} \text{Next}(r_j + p, \mathcal{C}(R_a, R_i)) - p & \text{if } |R_a| = n \\ \text{Next}(r_j + p, \mathcal{C}(R_a, R_i)) - p & \text{else if } S_j^{\min} \geq \\ & \quad \text{Next}(r_j + p, \\ & \quad \mathcal{C}(R_a, R_i)) - p \\ \text{Prev}(r_j + p, \mathcal{C}(R_a, R_i)) - p & \text{otherwise.} \end{cases}$$

4. *Use Theorem 2 to find an optimal schedule for instance* Π' *with finish times* $\mathcal{C}(R_a, R_i)$ *and let* $W^l_{R_a,R_i}$ *be the objective value of the solution of the assignment problem.*

We now show that Algorithm 1 finds a lower bound for the objective value of the optimal schedule among all (R_a, R_i)-schedules. In Theorem 2, we have shown that there is a one-to-one relation between the solution of the assignment problem and a schedule of MDS − EP. Let $\varsigma_l(R_a, R_i)$ be the schedule that can be constructed by the solution of the assignment problem in Algorithm 1.

Theorem 5 *Given a problem instance of* $1 \mid p_j = p, r_j \mid \sum \sum w_{j,k} U_{j,k}$ *and given two disjoint sets of times* R_a *and* R_i, *Algorithm 1 has the following properties.*

1. *If there is a feasible* (R_a, R_i)*-schedule, the algorithm terminates with a finite value.*
2. *If the algorithm terminates with a finite value, a job is started at each time in* R_a *and no job is started at a time in* R_i *in* $\varsigma_l(R_a, R_i)$.
3. *The solution is a lower bound* $W^l_{R_a,R_i}$ *for the objective value* $W^*_{R_a,R_i}$ *of the optimal* (R_a, R_i)*-schedule* $W^l_{R_a,R_i} \leq W^*_{R_a,R_i}$.

Note that an upper bound can be computed in a similar way. The only difference is that the release times are not adapted. Due to conditions 3 and 6 in Lemma 4, there is a feasible schedule with the given completion times.

Theorem 6 *Given a problem instance of* $1 \mid p_j = p, r_j \mid \sum \sum w_{j,k} U_{j,k}$ *and given two disjoint sets of times* R_a *and* R_i, *Algorithm 1 terminates in* $O(n^4)$.

Proof The proof is presented in the appendix. □

Note that the time complexity of the algorithm can be reduced to $O(n^3)$ by iterating simultaneously over $\mathcal{S}$, R_a, $R(\mathcal{J})$ and R_i, dividing the times into residue classes and keeping track if a job can start at a time of each residue class. However, defining this algorithm formally is omitted due to space limitations. Furthermore, the branching strategy described below adds at most $2n$ elements to R_i such that the complexity is also reduced.

The lower bound can be used to solve the problem with a branch-and-bound algorithm. We can use the following result to develop a branching strategy.

Theorem 7 *Given an instance of* MDS − EP *and two disjoint sets of start times* $R_a, R_i \in \mathcal{S}$, *let* $\hat{t} \in \mathcal{S}$ *be the minimum time such that* $\hat{t} \geq r_j$ *and* $(\hat{t} - \text{Prev}(r_j, R_a)) \mod p = 0$. *Suppose that for each* $t \in R(\mathcal{J})$ *with* $t > \text{Prev}(r_j, R_a)$ *and* $t < \hat{t}$ *it holds* $t \in R_i$. *Furthermore, suppose that it holds* $\hat{t} \in R_a \cup R_i$. *Then, job* J_j *is not started before its release date in* $\varsigma_l(R_a, R_i)$.

Proof The proof is presented in the appendix. □

We can use the previous results to construct a branch-and-bound algorithm. First start with a single node with two empty sets R_a and R_i and compute a lower bound solution for R_a and R_i. Either the schedule of the lower bound solution is feasible or there is a job J_j that starts before its release date. In the first case, terminate the algorithm. In the latter, create two child nodes. We want to find a schedule such that J_j is feasible. Therefore, add $t = \text{Prev}(r_{\hat{j}}, R(\mathcal{J}) \setminus (R_i \cup R_a))$ to R_a in one of the nodes and to R_i in the other node. If $\text{Prev}(r_{\hat{j}}, R(\mathcal{J}) \setminus (R_i \cup R_a))$ is smaller than $\text{Prev}(r_{\hat{j}}, R_a)$, add $r_{\hat{j}} + (\text{Prev}(r_{\hat{j}}, R_a) - r_{\hat{j}}) \mod p$ to the two nodes instead. With the previous result, we know that job $J_{\hat{j}}$ cannot start before its release date in the resulting schedule $\varsigma_l(R_a, R_i)$

as soon as all of these times have been added to R_a or R_i. After creating the two nodes, compute a lower bound for both of them with the new sets R_a and R_i. Then, create child nodes for the node with the lowest lower bound among all open nodes and repeat the process until a feasible solution is reached. Again the previous result ensures that only finitely many times have to be added to R_a or R_i until all jobs are feasible in the lower bound solution.

5 Equal processing time - Heuristics

As shown in the previous sections, the MDS − EP can be solved exactly by solving assignment problems. Either a single one if certain conditions are met or a combination of several assignment problems in a branch-and-bound algorithm for the $1 \mid p_j = p, r_j \mid \sum\sum w_{j,k}U_{j,k}$ problem. In the following, we outline some approaches how to solve the MDS − EP heuristically.

5.1 Heuristic algorithms

First we provide some greedy heuristics that can be used to find a solution to the MDS − EP problem with release dates. In the numerical analysis, we show results from the heuristics that are defined by the following rules.

Greedy algorithm without waiting: Start at time $t = 0$. Whenever there is at least one job J_j that has not yet been processed and with release date $r_j \leq t$, start the job with the highest sum of due date weights between $t + p$ and $t + 2p$ and with release date $r_j \leq t$. If there is no such job, set $t = \text{Next}(t, R(\mathcal{J}))$.

Greedy algorithm with waiting: Start at time $t = 0$. Whenever there is at least one job J_j that has not yet been processed and with release date, $r_j \leq t$ schedule the job j' such that the sum of weights of missed due dates between $t + p$ and $\max(t + 2p, r_{j'} + p)$ for all other unprocessed jobs with release date $r_{j'} < t + p$ is minimized. If there is no such job, set $t = \text{Next}(t, R(\mathcal{J}))$.

The two greedy algorithms always schedule the job that would increase the objective value if it was not scheduled assuming that it is otherwise processed directly after the next scheduled job. The waiting heuristic, however, may wait in case there are due dates for a job j with $d_{j,k} < t + 2p$.

Single assignment heuristic: Instead of solving multiple assignment problems in a branch-and-bound algorithm, one can restrict the problem with arbitrary release dates such that a job is started whenever the machine is idle and there is a job available. The finish times can therefore be chosen as

$$\begin{aligned} c_1 &= \min R(\mathcal{J}) + p \\ c_{i+1} &= \begin{cases} c_i + p, & \text{if } J(c_i) \geq i + 1 \\ \text{Next}(c_i, R_{\text{rel}}) + p, & \text{otherwise.} \end{cases} \end{aligned} \tag{5}$$

This constrained problem can again be solved by a single assignment problem, but the objective value may be worse than in the original problem. However, the worst-case optimality gap can be bounded as shown below. Note that the finish times of the greedy heuristic without waiting are the same as the finish times defined above. Since the assignment problem finds the optimal schedule with these finish times, the greedy heuristic without waiting can never result in a better schedule.

5.2 Optimality gap for the single assignment heuristic

Since we fix the finish times a priori, we use a slightly different notation from before. We write $w_j(t)$ for the sum of the weights of all missed due dates for job J_j at time t. Furthermore, we assume that the machine is never idle, i.e., for n jobs there is a feasible schedule with start times $0, p, \ldots, (n-1)p$ and with finish times $p, 2p, \ldots, np$. Since a job is scheduled whenever the machine is idle and a job is available, the only reason for an idle machine is that there are not enough jobs released. If this is the case, we can split the problem into multiple instances such all of them fulfill the assumption.

The bound for the optimality gap given below may be the trivial bound, for example if there is at most one due date for each interval between $\left[ip + 1, (i + 1)p\right]$ for all $i \leq n$. However, the more due dates of different jobs there are in each interval, the more bounding the following result is.

Theorem 8 *The worst-case optimality gap of the single assignment heuristic is smaller or equal to*

$$\sum_{k=1}^{n-1} \Delta_w^{max}(k) - \sum_{k=1}^{n-1} \Delta_w^{min}(k) \tag{6}$$

with

$$\Delta_w^{max}(k) = \max_{j \in J} \left\{ w_j((k+1)p) - w_j(kp) \right\}.$$

$\Delta_w^{min}(k)$ *is defined accordingly.*

Proof The proof can be found in the appendix. □

We can see that the optimality gap is dependent on the weight difference between two possible finish times. As stated before, in some instances the bound for the optimality gap is no strong result. In practice, however, the due dates are often defined in fixed intervals, e.g., at the end of a day, such that the worst-case optimality gap is lower the more jobs can be processed during one day. Denote by $\mathcal{D}$ the set of different due dates of all jobs. We assume that there are no two due dates of the same job at the same time. Then, we can follow the next result.

Corollary 5 *Let $w_j^{\max}$ be the maximum weight of all due dates of all jobs. If there is a minimum time $t^{\min} \geq p$ such that it holds for $d_i, d_j \in \mathcal{D}$ with $d_1 \neq d_2$*

$$|d_i - d_j| \geq t^{\min} \geq p,$$

the optimality gap of the single assignment heuristic can be bounded by

$$w_j^{\max} \left\lceil \frac{np}{t^{\min}} \right\rceil.$$

Proof The proof can be found in the appendix. □

6 Numerical results

In the following, we analyze the different solution approaches on random problem instances. One problem is generated by randomly creating n jobs with processing time p each. Each job has a release date between 0 and fnp with f being a fixed parameter for each instance. The larger the f is, the bigger is the average distance between the release dates, and therefore, the less jobs are waiting at the machine on average. Furthermore, each job has 10 due dates. The distance from the release date to the first due date is randomly generated between $b_{\min} p$ and $b_{\max} p$. The distance from one due date to the next due date is randomly selected between p and $5p$. The weights of the due dates are randomly generated between 1 and 10. We generate 20 problem instances for each combination of

- $n \in \{10, 15, 20, 25, 30, 35, 40\}$,
- $p \in \{1, 2, 5, 10, 20\}$,
- $f \in \{0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5\}$ and
- $(b_{\min}, b_{\max}) \in \{(1, 2), (3, 5), (6, 10)\}$.

Each instance is then solved by the branch-and-bound algorithm and the three heuristics from before. The numerical tests support the following arguments.

Waiting for a job j with a later release date instead of scheduling a job that is currently available has the positive effect that the job j can be scheduled earlier. In case the weights are high, this may improve the solution. Furthermore, if there are other jobs with later release dates and high weights, these jobs may also be scheduled earlier since it is not necessary to schedule job j anymore. However, as long as there are other jobs available to schedule, their finish times are increased and therefore miss on average more due dates if the machine is unnecessarily idle.

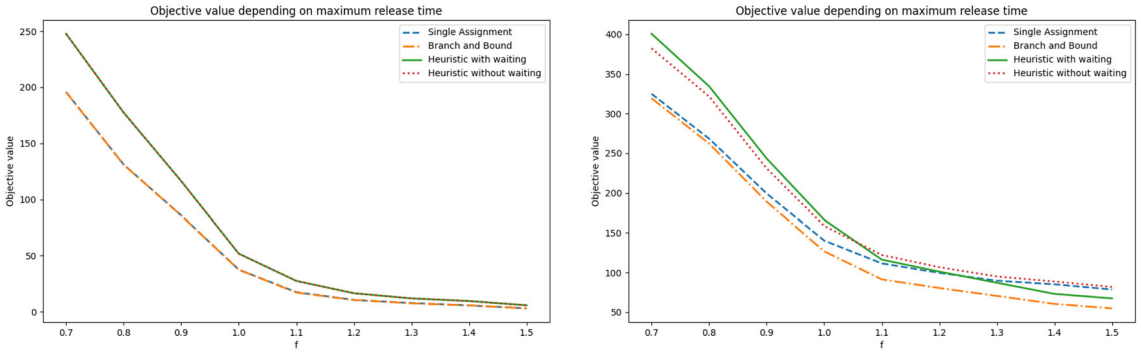
The parameters for Figs. 2, 3 and 4 are given in Table 1 with x being the parameter which has multiple values.

Influence of buffer times before the first due date: The first notable observation is that there is only a marginal difference in solution quality between the single assignment heuristic and the branch-and-bound algorithm in case the buffer between the release date and the first due date of each job is large enough. In Fig. 2a, the average objective value depending on the parameter f is depicted for the parameters given in Table 1. Note that lines for the single assignment heuristic and the branch and bound overlap, such that the one for the single assignment heuristic can be barely seen. The same is true for the two curves of the two heuristics with and without waiting.

The negative effect of waiting for a later job as described before is true for all buffer times. The positive effect of scheduling a job with high due date weights earlier, however, decreases since there are more possible finish times before the first due date. For random instances with large buffers, it is highly unlikely to require waiting for a later job in the optimal schedule. However, the bound of the optimality gap of Theorem 8 can only be improved slightly and worst-case instances can easily be constructed by hand. Also the branch-and-bound algorithm found the solution much faster in the problem instances with a bigger buffer.

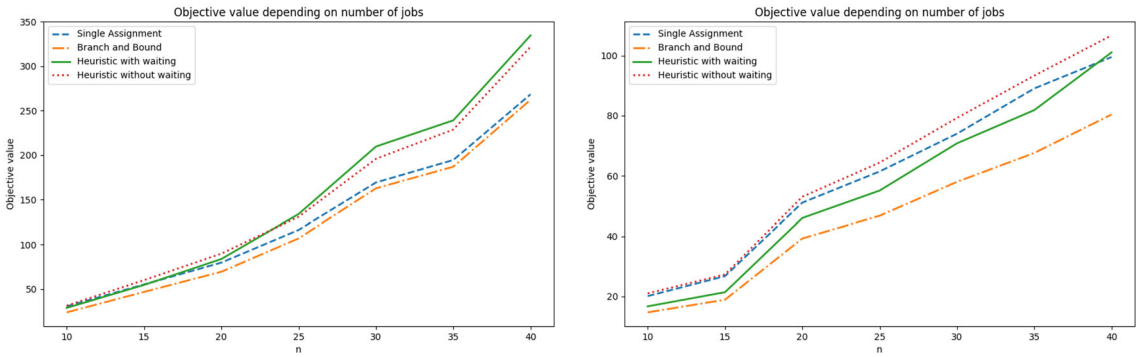
Comparing the two heuristics with waiting and without waiting, we can see that there is no difference for minimum buffer times bigger than $2p$. This can easily be explained since by construction the waiting heuristic may only wait for the release of job J_j in case job J_j would miss its first due date otherwise. Since there is enough buffer time between the release date and the first due date, this is not possible and the two heuristics are the same. In practice, there is often a minimum time between the release date and the first due date such that using the single assignment heuristic is promising. While a large buffer time is often enough to justify the usage of the single assignment heuristic, in theory the cases with low buffer times are more interesting since they show larger differences between the optimal value and the solution of the single assignment heuristic. Therefore, the following results fix the buffer time multiplier to $b_{\min} = 1$ and $b_{\max} = 2$.

Influence of the number of waiting jobs: The next observation is that the optimality gap between the single assignment heuristic and the optimal value is small in case the number of waiting jobs is large. Figure 2b shows that for low values of f there is only a small difference between the optimal value and the single assignment heuristic. Since on average more jobs arrive than can be processed by the machine, the number of jobs that are released but not processed grows on average before fnp for small f. Therefore, keeping the machine idle in order to wait for a job with later release date would increase the finish time of all these waiting jobs. Since less jobs are waiting on average for higher values of f, this effect decreases for high values of f and waiting becomes better more often. Therefore, the difference between the two



(a) High buffer times (b) Low buffer times

Fig. 2 Objective value depending on machine utilization



(a) High machine utilization (b) Low machine utilization

Fig. 3 Objective value depending on number of jobs

Fig. 4 Objective value depending on processing time

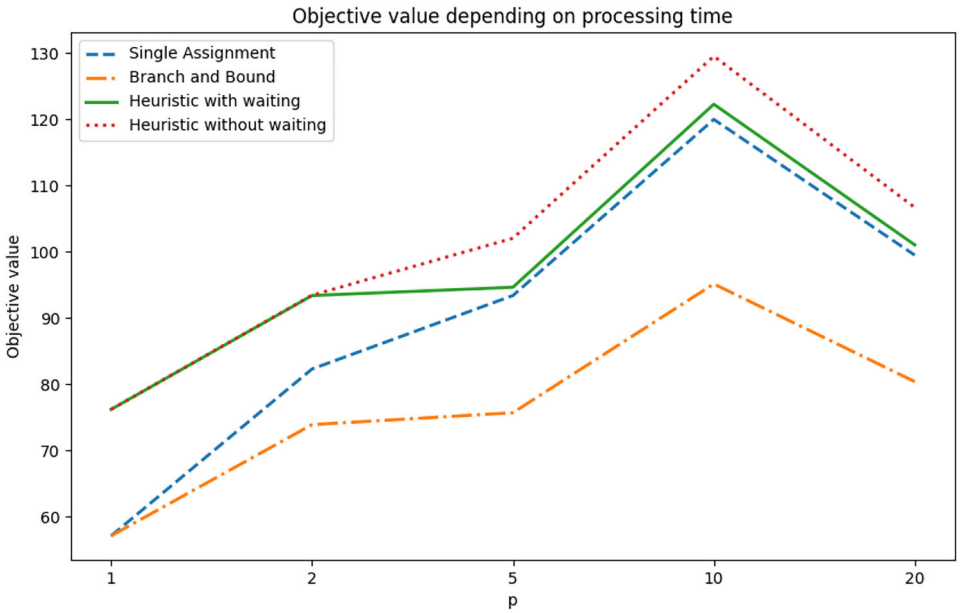


Table 1 Parameter list for numerical results

Figure	n	p	f	$b_{\min}$	$b_{\min}$
Fig. 2a	40	20	x	1	2
Fig. 2b	40	20	x	3	5
Fig. 3a	x	20	0.8	1	2
Fig. 3b	x	20	1.2	1	2
Fig. 4	40	x	1.2	1	2

solution approaches increases if the machine is not fully utilized.

Furthermore, while the single assignment heuristic finds a better solution than both greedy heuristics for low values of f, the greedy heuristic with waiting becomes better for high values of f since there are less waiting jobs, and therefore, waiting is better more often than not waiting. Note that the solution of the single assignment heuristic can never be worse than the heuristic without waiting since it finds the optimal schedule among all schedules that start a job whenever possible.

Influence of the number of jobs: In the numerical test, the optimality gap between the single assignment heuristic and the branch-and-bound algorithm is nearly constant for a growing number of jobs in case there is a high machine utilization. Figure 3b and a shows the average objective value depending on the number of jobs if the other parameters are fixed as given in Table 1.

If f is less than 1, more jobs arrive than can be processed. Therefore, the more jobs arrive in total the more jobs wait on average at the machine and waiting for a job with later release date increases the average completion time of all other unfinished jobs. On the other hand, if the machine is not fully utilized as in Fig. 3b there must be idle times between the jobs since not enough jobs arrive. Therefore, the average number of available jobs does not increase with an increasing number of total jobs. Because of these idle times, waiting for a later job only increases the finish time until the next idle time and the optimality gap of a single assignment heuristic increase with n.

Influence of the processing time: The optimality gap between the assignment problem and the branch-and-bound algorithm is larger for large processing times. In Fig. 4, the average objective value is depicted for different processing times with the other parameters fixed as given in Table 4.

We can see that the optimality gap of a single assignment heuristic is lower for small processing times. For $p = 1$, we have already shown that solving a single assignment problem is optimal. Furthermore, since waiting for a later jobs increases the finish time of other available jobs by at least $\frac{1}{p}$, the negative effect of waiting for a job with later processing time can be less for high processing times. Therefore, the difference between the objective values of the single assignment heuristic solution and the branch-and-bound solution increases for increasing p. Since the difference $\frac{1}{p}$ and $\frac{1}{p+1}$ goes to 0 for large p, this effect cannot be seen anymore for large p.

Computation times: To test the competitiveness of the branch-and-bound algorithm for the weighted tardiness problem, we generated problems as proposed in Gafarov et al. (2020). There, jobs are generated for different p with $r_j \in [0, (n-2)p]$, $d_j \in [0, (n-1)p]$, $w \in [1, 120]$. Furthermore, they require $w_1 \leq w_2 ... \leq w_n$ and $d_1 \leq d_2 ... \leq d_n$. Therefore, we first generated n due dates and weights and sorted them. Their data show that r_j may be bigger than d_j such that we did not restrict the release date to be smaller than the due date. In Gafarov et al. (2020), it is stated that the proposed algorithm did not solve instances for $n = 20$ in 60 min. The main focus of this work lies on the theoretical part. Therefore, we only implemented a non-optimized version of our algorithm in python and let the assignment problems be solved with the assignment solver of the SciPy library. We were able to solve 1000 instances with $p = 30$ and $n = 20$ in around 12 min. Implementing an optimized version of both algorithms and comparing their run time for different parameter go beyond the scope of this paper. However, the results show that the branch-and-bound algorithm is a promising candidate for a competitive algorithm for the weighted tardiness problem. Furthermore, there are ideas to improve the performance even further. For example, we branch at the first job of the lower bound solution that is not feasible in the original problem. Before doing that, one could check whether the position could be changed with another job without increasing the objective value. Doing so, one could reduce the unnecessary branches even further.

7 Relation to the weighted tardiness problem

In Theorem 1, we have used known results of the weighted tardiness problem, denoted by $1 \mid r_j, p_j = p \mid \sum w_j T_j$, to prove the NP-hardness of MDS. On the other hand, we can extend the results of the previous sections to the weighted tardiness problem. In both problems, the number of possible finish times is bounded by n^2 if the processing times are equal since the objective functions are regular, and therefore, an optimal left-shifted schedule exists. The weighted tardiness problem can be transformed into a problem with multiple due dates per job by defining a due date for each possible finish time. Therefore, we can extend the results from before to the weighted tardiness case as well.

Corollary 6 *The weighted tardiness problem* $1 \mid p_j = p, r_j \mid \sum w_j T_j$ *is fixed parameter tractable with the number of different release dates being fixed.*

Corollary 7 *The weighted tardiness problem* $1 \mid p_j = p, r_j \mid \sum w_j T_j$ *can be solved by a branch-and-bound algorithm by solving at most* 2^{N_r} *assignment problems.*

The optimality gap term of Theorem 8 can be adapted in the following way.

Corollary 8 *The optimality gap resulting from the single assignment heuristic is smaller or equal to*

$$p(n-1) \max_{J_j \in \mathcal{J}} w_j.$$

8 Summary and outlook

In this work, we introduced multiple due date scheduling MDS, a new scheduling model with multiple due dates per job, as an generalization to traditional due date scheduling problems. It can be used to find multipurpose algorithms since many problems with traditional objective functions, such as weighted number of late jobs, can be reduced to MDS. Concentrating on the one machine model, we showed that problem MDS is in general NP-hard in the strong sense and then focused on the special case where all processing times are equal, which we denoted by MDS − EP. We made this assumption since in the area of personalized medicine, often the same production steps with different input materials have to be performed.

We proved that problem MDS − EP can be solved exactly by solving either a single assignment problem or, in the presence of release dates, multiple assignment problems in a branch-and-bound algorithm. Furthermore, we showed that even in cases where solving a single assignment problem does not provide an optimal solution, the optimality gap can still be bounded.

Lastly, we provided several heuristics and compared them to each other, the heuristic to solve a single assignment problem and an algorithm for a similar problem using numerical experiments. In particular, the experiments showed that even though the theoretical optimality gap bound of solving just a single assignment problem is high, it performs well in practice as long as there is some buffer between the release date of a job and its first due date or if there is a high machine utilization. The numerical experiments also showed that the presented branch-and-bound algorithm is competitive with other state-of-the-art algorithms for solving instances of the weighted tardiness problem.

There are several possibilities for further research, e.g., algorithms for solving the MDS in a general flow shop environment are still missing. Furthermore, the solution approaches presented in this paper can be enhanced by finding an improved branching strategy for the branch-and-bound algorithm or by refining the look ahead of the greedy heuristics. Finally, this paper focused on the offline scenario of problem MDS, i.e., all jobs and their characteristics are known in advance. In real-world applications, however, it is often necessary to compute partial schedules without have complete knowledge about future job arrivals. Therefore, in the future it would be interesting to consider online versions of problems MDS, where schedules have to be computed without complete knowledge of the problem instance in advance.

Appendix A Proof of Theorem 1

Proof In what follows, we denote by $\mathcal{D}(\mathcal{P}, K)$ the decision version of some scheduling problem $\mathcal{P}$, i.e., the problem of deciding, given an instance Π of problem $\mathcal{P}$, does there exists a schedule σ for instance Π with objective value no larger than K for some given $K \in \mathbb{N}$. To show the NP-hardness, we show that problem $1 \mid\mid \sum w_j T_j$ can be pseudo-polynomially reduced to $1 \mid\mid \sum w_{j,k} U_{j,k}$. Given an instance Π of the decision problem $\mathcal{D}(1 \mid\mid \sum w_j T_j, K)$, we define an instance Π' of $\mathcal{D}(1 \mid\mid \sum w_{j,k} U_{j,k}, K)$. For each job J_j in problem instance Π, we define a job J'_j for problem instance Π' and set $p'_j = p_j$. Additionally we define

$$c_{\max} = \sum_{j=1}^{n} p_j.$$

For each job J_j with due date d_j in Π and for each time $t \in \{d_j, d_j + 1, ..., c_{\max} + 1\}$, we define a due date $d'_{i,j}$ in Π' with weight $w'_{i,j} = w_j$. We write $W(\varsigma)$ for the objective value of a schedule ς in Π and $W'(\varsigma)$ for the objective value of a schedule ς in Π'.

We first show that if there is a solution for instance Π with objective value no larger than K, it follows that there is also a solution to Π' with objective value no larger than K. Let ς be a schedule with objective value K in Π. Since the processing times are the same for all jobs in instances Π and Π', the schedule $\varsigma' = \varsigma$, where each job J'_j of instance Π' has the same start and completion time in ς' as job J_j in ς, is feasible for instance Π'. The objective value of schedule ς' is

$$\begin{aligned} W'(\varsigma') &= \sum_{j=1}^{n} \sum_{k=1}^{K_j} w'_{j,k} U_{j,k}(\varsigma') \\ &= \sum_{j=1}^{n} \sum_{k=1}^{K_j} w_j U_{j,k}(\varsigma') \\ &= \sum_{j=1}^{n} w_j (\max\{C_j(\varsigma') - D_j, 0\}) \end{aligned}$$

$$= W(\varsigma)$$
$$\leq K.$$

Now we show the opposite direction. Suppose there exists a schedule ς' for instance Π' with objective value no larger than K. Then the schedule $\varsigma = \varsigma'$, where each job J_j has the same start and completion time in ς as job J'_j has in ς', is feasible for instance Π and has objective value

$$\begin{aligned} W(\varsigma) &= \sum_{j=1}^{n} w_j(\max\{C_j(\varsigma) - D_j, 0\}) \\ &= \sum_{j=1}^{n}\sum_{k=1}^{K_j} w_j U_{j,k}(\varsigma') \\ &= \sum_{j=1}^{n}\sum_{k=1}^{K_j} w'_{j,k} U_{j,k}(\varsigma') \\ &= W'(\varsigma') \\ &\leq K. \end{aligned}$$

Therefore, there is a schedule ς for instance Π with objective value no larger than K if and only if there is a schedule ς' for instance Π' with objective value no larger than K. Thus, the above construction reduces problem $\mathcal{D}(1 \mid\mid \sum w_j T_j, K)$ to problem $\mathcal{D}(1 \mid\mid \sum w_{j,k} U_{j,k}, K)$. Since the constructed reduction keeps the number of jobs as well as their processing times and weights exactly the same and adds at most $n \cdot c_{\max}$ due dates, the reduction is pseudo-polynomial. Since the decision version of problem $1 \mid\mid \sum w_j T_j$ is NP-complete in the strong sense and the decision version of problem $1 \mid\mid \sum w_{j,k} U_{j,k}$ is in NP (for a given schedule it can be decided in at most $O(n^2)$ time if it is feasible and its objective value is no larger than K), the given pseudo-polynomial reduction implies that the decision version of problem $1 \mid\mid \sum w_{j,k} U_{j,k}$ is NP-complete in the strong sense (cf. Garey and Johnson (1979)) and problem $1 \mid\mid \sum w_{j,k} U_{j,k}$ is NP-hard in the strong sense. □

Appendix B Proof of Theorem 3

Proof Proof of Claim 1: Assume that $c < \infty$ for each $c \in \mathcal{C}(R_{\text{rel}})$ and $r + p \in \mathcal{C}(R_{\text{rel}})$ for each $r \in R_{\text{rel}}$. Let $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ be the set of jobs and assume that $r_j \leq r_{j+1}$ for each $1 \leq j \leq n-1$ (otherwise simply renumber the jobs). We show that schedule ς constructed by setting $S(J_j) = c_j - p$ and $C(J_j) = c_j$ for each $1 \leq j \leq n$ is a feasible R_{rel}-schedule. First, note that since $c_j < \infty$ for all $c_j \in \mathcal{C}(R_{\text{rel}})$, by definition of $\mathcal{C}(R_{\text{rel}})$ and due to $\text{Next}(c_j, R_{\text{rel}}) \geq c_j$ it holds $c_{j+1} \geq c_j + p$ for each $1 \leq j \leq n-1$. Thus, by definition of ς and because the processing time of each job is equal to p, no two jobs overlap in ς.

To prove feasibility of ς, we are left to show that no job starts before its own release date in ς. For job J_1, this is clear by definition of completion time c_1. Suppose now that job J_j is not started before its own release date in schedule ς. We show that job J_{j+1} is not started before its own release date in ς. Indeed, first assume that $c_{j+1} = c_j + p$. Then by definition of c_{j+1} at least $j+1$ jobs are released at time c_j. This means that job J_{j+1} was released before or at time c_j (due to the chosen job numbering), and since it is started at time c_j, it is not started before its own release date. Otherwise, we have $c_{j+1} = \text{Next}(c_j, R_{\text{rel}}) + p$. Then, since $c_{j+1} < \infty$ by assumption, $\text{Next}(c_j, R_{\text{rel}})$ is finite and therefore is the release date of some job $J_k \in \mathcal{J}$ by definition of R_{rel}. Furthermore, since job J_j finishes at time c_j and is not started before its own release date, this means that job J_k is released after job J_j, i.e., $r_j < r_k$. Again by the chosen numbering of the jobs, J_{j+1} is the first job released after J_j (though the two jobs may also be released at the same time). Thus, for the release date r_{j+1} of job J_{j+1} it holds that $r_{j+1} \leq \text{Next}(c_j, R_{\text{rel}})$, and thus, again job J_{j+1} is not started before its own release date in ς. By induction, it follows that in ς no job is started before its own release date.

To finish the proof, we are left to show that ς is a R_{rel}-schedule. For this, note that for each $r \in R_{\text{rel}}$ it holds $r + p \in \mathcal{C}(R_{\text{rel}})$, and because the processing time of each job is equal to p, for each $r \in R_{\text{rel}}$ a job is started at time r in ς. Finally, note that $r + p \notin \mathcal{C}(R_{\text{rel}})$ for any $r \in R(\mathcal{J})\backslash R_{\text{rel}}$. Indeed, due to the definition of $\mathcal{C}(R_{\text{rel}})$ for any $c_j \in \mathcal{C}(R_{\text{rel}})$ either $c_j - p \in R_{\text{rel}}$ (cf. the definition of c_1 and the second case of the definition of c_{j+1} in Eq. (4)), or $c_j = c_{j-1} + p$ and $c_{j-1} \notin R(\mathcal{J})\backslash R_{\text{rel}}$ (cf. the first case of the definition of c_{j+1} in Eq. (4)). Therefore, in ς no job is started at any time $r \in R(\mathcal{J})\backslash R_{\text{rel}}$. Thus, ς is both feasible and an R_{rel}-schedule, which finishes this part of the prove.

Proof of Claim 2: The statement is proved via induction. Since $c_1 = \min\{r \in R_{\text{rel}}\} + p$ at least one job is released at time $c_1 - p$. Suppose that the claim holds for $i < n$. Then, in case $c_{i+1} = c_i + p$ by the first part of Eq. (4), it follows $J(c_i) \geq i + 1$ (by the condition on that part of Eq. (4)), and at least $i + 1$ jobs are released at time $c_{i+1} - p = c_i$. Otherwise, we have $c_{i+1} = \text{Next}(c_i, R_{\text{rel}}) + p$. In that case, if there exists some $r \in R_{\text{rel}}$ such that $c_i \leq r$ it follows $c_{i+1} = r + p$. r is the release date of some job, and i jobs are released by time $c_i - p$ by assumption that means at time $c_{i+1} - p = r$ at least $i + 1$ jobs are released. Lastly, if no such r exists, we have $c_{i+1} = \infty$. In that case, since the release dates of all jobs are finite $n \geq i + 1$ jobs are released by time c_{i+1}, which finishes the proof.

Proof of Claim 3: For this proof, we use notation $R_{\text{rel}} = \{r^1, r^2, \ldots, r^\ell\}$ with $r^i \leq r^j$ for all $i \leq j$.

$\Rightarrow$: First, we show that $|r^i - r^j| \geq p$ for all $r^i, r^j \in R_{\text{rel}}$ with $i \neq j$ implies $r^i + p \in \mathcal{C}(R_{\text{rel}})$ for all $r^i \in R_{\text{rel}}$. Assume otherwise, and let $r^i \in R_{\text{rel}}$ be the minimum time, such that $r^i + p \notin \mathcal{C}(R_{\text{rel}})$. First, note that by Eq. (4) we have $c_1 = r^1 + p$, and therefore, $i \geq 2$. Additionally, note that $r^2 \geq r^1 + p$ by assumption, and thus, $c_1 \leq r^2 \leq r^i$. Let $c_k \in \mathcal{C}(R_{\text{rel}})$ be the maximum time such that $c_k \leq r^i$. Observe that $k < n$. Indeed, since at time $r^i \geq c_k$ a job is released, there can be at most $n-1$ jobs released at time $c_k - p$. Thus, by Claim 2 of the Lemma, we have $k \leq n-1$. Now, since $k < n$ there exists $c_{k+1} \in \mathcal{C}(R_{\text{rel}})$ with $c_{k+1} > r^i$ by choice of c_k and construction of $\mathcal{C}(R_{\text{rel}})$. First, assume that $c_{k+1} = c_k + p$ with $c_{k+1} \leq \text{Next}(c_k, R_{\text{rel}})$. Then, since $r^i \geq c_k$ and thus $r^i \geq \text{Next}(c_k, R_{\text{rel}})$ it follows that $c_{k+1} \leq r^i$, a contradiction to the choice of c_k. Otherwise, by Eq. (4), we have $c_{k+1} = \text{Next}(c_k, R_{\text{rel}}) + p$. Since $r^i + p \notin \mathcal{C}(R_{\text{rel}})$, it follows that $r^i \neq \text{Next}(c_k, R_{\text{rel}})$, and since $r^i \geq c_k$, it follows further that $r = \text{Next}(c_k, R_{\text{rel}}) < r^i$. However, since $r^i - r \geq p$ by assumption and $c_{k+1} = r + p$ by construction, it follows that $c_{k+1} \leq r^i$, again a contradiction to the choice of c_k.

$\Leftarrow$: To prove the other direction, suppose that $r^i + p \in \mathcal{C}(R_{\text{rel}})$ for all $r^i \in R_{\text{rel}}$. By construction, it holds $c_{k+1} \geq c_k + p$. Therefore, for each $r^i \in R_{\text{rel}}$ with $i < \ell$ we have $r^i + p + p \leq r^{i+1} + p$ which implies $r^i + p \leq r^{i+1}$. Thus, $r^j - r^i \geq p$ for all $i < j$, which finishes the proof.

Proof of Claim 4: In order to show the claim, we first show the following result. Let $R^1_{\text{rel}}, R^2_{\text{rel}} \subseteq R(\mathcal{J})$ be two sets of release dates and let c^1_i be the i-th time in $\mathcal{C}(R^1_{\text{rel}})$ and c^2_i be the i-th time in $\mathcal{C}(R^2_{\text{rel}})$. If there is a $r^* \in R^1_{\text{rel}} \cap R^2_{\text{rel}}$ with $\{r \in R^1_{\text{rel}} | r \leq r^*\} = \{r \in R^2_{\text{rel}} | r \leq r^*\}$ and $c^1_i \leq r^*$, it holds $c^2_i = c^1_i$.

First suppose $i = 1$. Since $c_i \leq r^*$ and $\{r \in R^1_{\text{rel}} | r \leq r^*\} = \{r \in R^2_{\text{rel}} | r \leq r^*\}$, it follows that $c^1_1 - p = \min\{r \in R^1_{\text{rel}}\} = \min\{r \in R^2_{\text{rel}}\} = c^2_1 - p$, and therefore, the claim is true.

Now suppose $i > 1$ and that the claim holds for $i-1$, i.e., we have $c^1_{i-1} = c^2_{i-1}$. First, again since $\{r \in R^1_{\text{rel}} | r \leq r^*\} = \{r \in R^2_{\text{rel}} | r \leq r^*\}$, $c^1_{i-1} = c^2_{i-1}$ and $c_i \leq r^*$ by assumption, it holds that $c_{i-1} \notin R(\mathcal{J}) \setminus R^1_{\text{rel}}$ if and only if $c_{i-1} \notin R(\mathcal{J}) \setminus R^2_{\text{rel}}$. Furthermore, we have $\text{Next}(c_{i-1}, R^1_{\text{rel}}) = \text{Next}(c_{i-1}, R^2_{\text{rel}}) \leq r^*$. Finally we have $J(c^1_{i-1}) = J(c^2_{i-1})$. Thus, it follows by Eq. (4) that $c^1_i = c^1_{i-1} + p$ if and only if $c^2_i = c^2_{i-1} + p$ and $c^1_i = \text{Next}(c^1_{i-1}, R^1_{\text{rel}})$ if and only if $c^2_i = \text{Next}(c^2_{i-1}, R^2_{\text{rel}})$. Again, using $c^1_{i-1} = c^2_{i-1}$ and $\text{Next}(c_{i-1}, R^1_{\text{rel}}) = \text{Next}(c_{i-1}, R^2_{\text{rel}}) \leq r^*$, it follows that $c^1_i = c^2_i$.

We now proof the claim of the lemma by showing that there exists some set of release dates $R_{\text{rel}} \subseteq R(\mathcal{J})$ such that there exists an optimal, left-shifted R_{rel}-schedule with the first i completion times being equal to the first i times in $\mathcal{C}(R_{\text{rel}})$ for $i = 1, ..., n$. Note that the statement is the same as the claim in the Lemma if $i = n$.

We first consider the case $i = 1$. Since the objective function is regular, there exists an optimal left-shifted schedule. Let ς^1 be any of these schedules and let $R^1_{\text{rel}} \subseteq R(\mathcal{J})$ be the set of release dates at which jobs start in ς^1. By definition of R^1_{rel}, the first job in ς_1 must start at $\min\{r \in R^1_{\text{rel}}\}$ and therefore finish at $\min\{r \in R^1_{\text{rel}}\} + p$. By definition of $\mathcal{C}(R^1_{\text{rel}})$, it holds $c_1 = \min\{r \in R^1_{\text{rel}}\} + p$, and therefore, there is a set $R^1_{\text{rel}} \subseteq R(\mathcal{J})$ and an optimal left-shifted schedule such that the first completion time is equal to the first time in $\mathcal{C}(R^1_{\text{rel}})$. Now suppose the claim is true for $i < n$. We show that the claim also holds for $i+1$. Let R^i_{rel} be the set of release dates such that there is an optimal left-shifted schedule ς^i with the first i completion times being equal to the first i times in $\mathcal{C}(R^i_{\text{rel}})$. If the $(i+1)$th finish time in ς^i is equal to the $(i+1)$th finish time in $\mathcal{C}(R^i_{\text{rel}})$, the claim is true for $i+1$ with $\mathcal{C}(R^{i+1}_{\text{rel}}) = \mathcal{C}(R^i_{\text{rel}})$.

Now suppose otherwise. We consider the two cases $c_{i+1} = c_i + p$ and $c_{i+1} \neq c_i + p$ for c_{i+1} and c_i as defined by $\mathcal{C}(R^i_{\text{rel}})$ with Eqn. (4). First assume $c_{i+1} = c_i + p$. Note that in left-shifted R_{rel}-schedules jobs must either start at the finish time of other jobs or at a time in R_{rel}. Since the $(i+1)$th finish time in ς^i is not equal to c_{i+1}, the $(i+1)$th finish time in ς^i must be equal to $\text{Next}(c_i, R^i_{\text{rel}}) + p$. By the conditions of Eq. (4), it must hold that $\text{Next}(c_i, R^i_{\text{rel}}) \geq c_i + p$ and $J(c_i) \geq i+1$. Therefore, there must be at least one job J_k with release date smaller or equal than c_i and that is started after c_i in ς_i. Since there is no job processed between c_i and $\text{Next}(c_i, R^i_{\text{rel}})$ and $\text{Next}(c_i, R^i_{\text{rel}}) - c_i \geq p$, the job J_k could be started at c_i as well. We now construct another set of release dates R^{i+1}_{rel} and an optimal left-shifted R^{i+1}_{rel}-schedule such that the first $i+1$ completion times are equal to the first $i+1$ times of $\mathcal{C}(R^{i+1}_{\text{rel}})$.

1. Set $\varsigma^{i+1} = \varsigma^i$.
2. Let job J_k start at c_i, i.e., $S_k(\varsigma^{i+1}) = c_i$.
3. Shift the remaining jobs as early as possible, i.e., for $l = i+2, \ldots, n$, let the job in position l start at the maximum of its release date and the completion time of the job in position $l-1$.
4. Let R^{i+1}_{rel} be the set of release dates, at which jobs start in ς^{i+1}.

Note that the first i jobs in ς^{i+1} are equal to the first i jobs in ς^i. Furthermore, job J_k starts at c_i. Therefore, the first $i+1$ completion times in ς^{i+1} are equal to the first $i+1$ times in $\mathcal{C}(R^i_{\text{rel}})$. We define $\hat{r}$ to be the time at which the job in the $(i+1)$-position in ς^i is started. Since it is not a completion time of another job and the schedule is left-shifted, it must be the release date of this job. Since job J_k is not started at a release date, it holds $\{r \in R^i_{\text{rel}} | r \leq \hat{r}\} = \{r \in R^{i+1}_{\text{rel}} | r \leq \hat{r}\}$. Therefore, due to result above, the construction of the first $i+1$ finish times is the same for both $\mathcal{C}(R^i_{\text{rel}})$ and $\mathcal{C}(R^{i+1}_{\text{rel}})$.

Furthermore, ς^{i+1} is a left-shifted schedule by construction. Therefore, R_{rel}^{i+1} and $\mathcal{C}(R_{\text{rel}}^{i+1})$ fulfill the claim for $i+1$.
Now suppose that $c_{i+1} \neq c_i + p$. Then, it must hold that $c_{i+1} = \text{Next}(c_i, R_{\text{rel}}^i)$. In a left-shifted R_{rel}-schedule, jobs may only start at their release dates or at completion times of other jobs. Since the first i jobs in ς^i are equal to the first i times in $\mathcal{C}(R_{\text{rel}}^i)$ and since the $(i+1)$th job in ς^i is not equal to $c_{i+1} = \text{Next}(c_i, R_{\text{rel}}^i)$, the $(i+1)$th job in ς^i must start at c_i. Due to $c_{i+1} = \text{Next}(c_i, R_{\text{rel}}^i)$, one of the conditions $c_i \notin R(\mathcal{J})\backslash R_{\text{rel}}^i$, $J(c_i) \geq i+1$ and $c_i + p \leq \text{Next}(c_i, R_{\text{rel}}^i)$ must be violated by ς^i. Therefore, schedule ς^i must either be infeasible or cannot be an R_{rel}-schedule. This is a contradiction to the assumption, which finishes the proof. □

Appendix C Proof of Lemma 2

Proof 1. By definition, no job is allowed to start at any $t \in R_{\text{i}}$ and $t_{\text{act}}^{\max}(\varsigma, J_j)$ must be a release date or in R_{a}.
2. For any feasible $(R_{\text{a}}, R_{\text{i}})$-schedule, a job must start at $\text{Prev}(r_j, R_{\text{a}})$. Furthermore, $\text{Prev}(r_j, R_{\text{a}})$ is less or equal to r_j. Since $t_{\text{act}}^{\max}(\varsigma, J_j)$ is the maximum of all release dates or times in R_{a} at which a job starts, property 2 must hold.
3. In a $(R_{\text{a}}, R_{\text{i}})$-schedule, all jobs must start at a release date, a time in R_{a} or at the completion time of a job that is not in R_{i}. Since $t_{\text{act}}^{\max}(\varsigma, J_j)$ is the maximum time before S_j that is either a release date or a time in R_{a}, a job has to start for each t with $t_{\text{act}}^{\max}(\varsigma, J_j) \leq t \leq S_j$ and $(t_i - t_{\text{act}}^{\max}(\varsigma, J_j)) \mod p = 0$. Since no job is allowed to start for any time in R_{i}, condition 3 must be true.

□

Appendix D Proof of Lemma 3

Proof Let ς be any feasible schedule. Assume that t' is the maximum time in ς before or equal to S_j at which a jobs starts and that is either a release date or a time in R_{a}. Therefore, t' must be in $T_{\text{act}}^{\max}(J_j)$ and $S_j \geq t'$. Furthermore, the job must start after its release date and the difference between t' and S_j must be a multiple of p since only all jobs in between must start at the completion time of another job. Therefore, we can conclude $S_j(\varsigma) \geq \max(t, r_j + (r_j - t') \mod p)$. Since

$$\min_{t \in T_{\text{act}}^{\max}(J_j)} (\max(t, r_j + (t - r_j) \mod p)) \leq \max(t', r_j + (r_j - t') \mod p),$$

the claim follows directly. □

Appendix E Proof of Lemma 4

Proof 1. The definition of $(R_{\text{a}}, R_{\text{i}})$-schedules requires all jobs to start at times in $\mathcal{S}$.
2. The first job cannot start before the first time in R_{a} or the first release date that is not in R_{i}.
3. If the job at position i cannot start before t, the job at position $i+1$ cannot start before $t+p$ since only one job can be processed at a time.
4. If t is in R_{a} a job must finish at $t+p$. Since no two jobs can be processed at the same time, no other job can finish between t and $t+p$.
5. Since at all times in R_{a} a job must be started, there can be at most $n-i$ times in R_{a} which are larger than c_i.
6. If the number of release dates smaller than $c_i^* - p$ is less than i, and $i-1$ jobs have already been scheduled, no job could be started at $c_i^* - p$.
7. The same arguments as in Lemma 2 ensure that for a job to start at time t there must be a release date t_a or a time in R_{a} such that whenever a job finishes between t_a and t another job must be started. This time can therefore not be in R_{i}.

□

Appendix F Proof of Theorem 5

Proof Proof of Claim 1: Let ς be a feasible $(R_{\text{a}}, R_{\text{i}})$-schedule. Then, it must hold that $|t_1 - t_2| \geq p$ for $t_1, t_2 \in R_{\text{a}}$ and $t_1 \neq t_2$ since only one job can be processed at a time and at all times in R_{a} a job must start. Furthermore, it must hold $J(t) \geq |\{t' \in R_{\text{a}} | t' \leq t\}|$ for any $t \in R_{\text{a}}$. Assume otherwise. Then, there is a t with $J(t) < |\{t' \in R_{\text{a}} | t' \leq t\}|$. Since at all times in R_{a} a job must start and only $J(t)$ jobs are released at time t, at least one job must start at a time, when it is not released. This is a contradiction, and therefore, the first step of the algorithm does not terminate in the first step if there is a feasible $(R_{\text{a}}, R_{\text{i}})$-schedule.
Therefore, it remains to show that the algorithm constructs n finish times in step 2 and that the solution of the assignment problem with these n finish times is finite. Let $C(\varsigma)_i$ be the finish time of the job in the i-th position in ς. We show for $i = 1, ..., n$ that at least i finish times are constructed by the algorithm and $c_i \leq C(\varsigma)_i$. We first show the claim for $i = 1$. Assume that the claim is not true. Then, it must hold $C(\varsigma)_1 < c_1$. Since jobs cannot start before their release dates in a feasible schedule, it must hold $C(\varsigma)_1 \geq \min(R(\mathcal{J})) + p$. Since it holds $C(\varsigma)_1 < c_1$, it must hold that $c_1 \neq \min(R(\mathcal{J})) + p$, and therefore, either $\min(R(\mathcal{J})) + p \leq \min(R_{\text{a}})$ or $|R_{\text{a}}| = n$. In the first case, no job can start at $\min(R(\mathcal{J}))$ since only one job can be processed at a time and a job must start at $\min(R_{\text{a}})$. In the second case, a job must start at all times in R_{a}. Since there are

n times in R_a, no job can start at a time not in R_a. Therefore, the claim holds for $i = 1$.

We now show that the claim also holds for $1 < i \leq n$ if it holds for $i-1$. Therefore, we assume that there is a schedule with $i-1$ finish times and $c_{i-1} \leq C(\varsigma)_{i-1}$. The algorithm iterates over all times in $\mathcal{S}$. If it holds $c_i \leq C(\varsigma)_i$, the claim is shown for i. Otherwise, the algorithm either terminate with an infinite lower bound or with $c_i > C(\varsigma)_i$. In both cases, the algorithm iterated over $C(\varsigma)_i - p$ as a potential start time for the i-th job since $c_{i-1} \leq C(\varsigma)_{i-1} \leq C(\varsigma)_i - p$. Since ς is a feasible schedule, the start time $C(\varsigma)_i - p$ fulfills all conditions of Lemma 4 with t_{act} being the largest release date or time in R_a at which a job starts in ς and that is smaller than c_i. Therefore, the algorithm would have selected $c_i = C(\varsigma)_i$ which is a contradiction. Therefore, the claim also holds for i. Since it also holds for $i = 1$, we can conclude by induction that the claim must be true for $i = n$.

It remains to show that the solution of the assignment problem is finite. In Theorem 2, we have shown the connection between the solution of the assignment problem and the objective value of an associated schedule. Since the finish times are separated by at least p, it is enough to show that there is a schedule with the constructed finish times such that no job is started before its release date. Condition 6 ensures that at least i jobs are released at time $c_i - p$. Therefore, if the job with the i-lowest release date is started at $c_i - p$, no job is started before its own release date.

Proof of Claim 2: Since the algorithm terminates with a finite value, we can assume that no $t_1, t_2 \in R_a$ exist with $t_1 \neq t_2$ and $|t_1, t_2| < p$. Additionally, we can assume that

$$J(t) \geq |\{t' \in R_a | t' \leq t\}|$$

for any $t \in R_a$ and that n completion times have been constructed. We show by contradiction that a job starts at each time in R_a. Assume otherwise and let t be the minimum time in R_a such that no job starts at t. Denote by $\hat{i}$ the maximum index such that $c_{\hat{i}} \leq t$. If no such $c_{\hat{i}}$ exists, it must hold that $c_1 > t + p$. Since t is in R_a and by the assumptions above, $t + p$ fulfills all conditions of Lemma 4. This is a contradiction, since c_1 is chosen by the algorithm instead.

Therefore, we assume that there is such an $\hat{i}$. The condition $|\{t \in R_a | t \geq c_{\hat{i}}^*\}| \leq n - i$ of Lemma 4 ensures that $\hat{i} < n$. Since t is in R_a, the condition 4 of Lemma 4 and the fact that there are no two times in R_a with distance smaller than p ensure that no job finishes at a time t' with $t < t' < t + p$. Since t is in R_a and $c_{\hat{i}}$ fulfills all conditions of Lemma 4, $t + p$ also fulfills the conditions with $t_a = t$. Therefore, $t + p$ would be chosen by the algorithm as $c_{\hat{i}+1}$ such that a job starts at t. This is a contradiction to the assumption and at all times in R_a a job must start.

Finally, suppose that in $\varsigma_l(R_a, R_i)$ a job is started at a time $t_i \in R_i$. In $\varsigma_l(R_a, R_i)$, jobs start only at times such that $t_i + p \in \mathcal{C}(R_a, R_i)$. By construction of c_1, it follows directly that $c_1 - p \notin R_i$. Therefore, $t_i + p = c_i$ with $i > 1$. Then, there must be a $t_{\text{act}} \in (R(\mathcal{J}) \cup R_a) \backslash R_i$ with $t_{\text{act}} \geq \text{Prev}(t_i, R_a)$, $t_{\text{act}} \leq t_i$ and $(t_{\text{act}} - t_i) \mod p = 0$. Due to $(t_{\text{act}} - t_i) \mod p = 0$, the algorithm would continue with the next s in step 2C. This is a contradiction to $t_i = c_i$.

Proof of Claim 3: If there is no feasible schedule, any value is a lower bound and nothing is to show. Therefore, we assume that there is a feasible schedule. We have shown before that the algorithm terminates with a finite value and that a job is started at each time in R_a and no time in R_i.

We show that any schedule feasible in Π is also feasible in Π', and therefore, the objective value of the optimal (R_a, R_i)-schedule in Π' is not higher than the objective value of the optimal (R_a, R_i)-schedule in Π. We then show that the assignment problem solves Π' optimally such that its solution value is a lower bound for the objective value of the optimal (R_a, R_i)-schedule in Π.

We first add a constraint to Π' such that only schedules are feasible that fulfill the conditions of Lemma 4. Note that any feasible schedule in Π must fulfill these conditions, such that it remains feasible in Π'. However, by relaxing other constraints later, this constraint leads to a higher lower bound.

Lemma 3 ensures that job J_j cannot start before $S_j^{\min}$. By construction, its release date is only increased to

$$\text{Next}(r_j + p, \mathcal{C}(R_a, R_i)) - p$$

if $|R_a| = n$ or if

$$S_j^{\min} \geq \text{Next}(r_j + p, \mathcal{C}(R_a, R_i)) - p.$$

In the first case, $\mathcal{C}(R_a, R_i)$ is equal to R_a due to the previous claim. Since in any feasible (R_a, R_i)-schedule a job must start for each time in R_a, any feasible schedule must use exactly the completion times defined by $\mathcal{C}(R_a, R_i)$. Therefore, if there is a job with $r_j + p \notin \mathcal{C}(R_a, R_i)$, its earliest start time in any feasible (R_a, R_i)-schedule is $\text{Next}(r_j + p, \mathcal{C}(R_a, R_i)) - p$. In the second case, we know with Lemma 3 that job J_j cannot start before $S_j^{\min}$. Therefore, all feasible schedules in Π remain feasible. The same holds true for any job with reduced release date. Therefore, any schedule feasible in Π is feasible in Π' as well, and since the due dates are not adapted, the objective values are the same in both instances.

Next we show that there is an optimal schedule for the adapted problem Π' with finish times defined as $\mathcal{C}(R_a, R_i) = \{c_1, \ldots, c_n\}$. Then, there must be an $i \leq n$ such that there is an optimal schedule with the first $i-1$ completion times equal to the first $i-1$ completion times of $\mathcal{C}(R_a, R_i)$ but not with the first i completion times. Let ς^* be an optimal schedule with finish times $\{c_1^*, \ldots, c_n^*\}$ and $c_k^* = c_k$ for $k \leq i$. The algorithm only continues with the next s, if a condition of

Lemma 4 is violated. Since these conditions must be met by any feasible schedule, it must hold $c_i^* > c_i$.

We first check $i = n$. By the third condition of lemma 4, all n jobs must be released at $c_n - p$. Therefore, the job that finishes at c_i^* in the optimal schedule could also be started at $c_i - p$. Due to the regularity of the objective function, this would result in an optimal schedule as well. Therefore, there would be an optimal schedule with finish times defined by $\mathcal{C}(R_\mathrm{a}, R_\mathrm{i})$.

Therefore, we assume $i < n$ and check $c_i < c_i^* < c_{i+1}$. Since by construction all jobs have a release date r'_j such that $r'_j + p$ is in $\mathcal{C}(R_\mathrm{a}, R_\mathrm{i})$, the release date of the job that finishes at c_i^* must be smaller or equal than $c_i - p$. Since i is the smallest i such that $c_i^* \neq c_i$, it holds $i = 1$ or $c_{i-1} = c_{i-1}^*$ and the job that finishes at c_i^* could also be started at $c_n - p < c_i^* - p$. Since the objective function is regular, this would not increase the objective value and one could construct an optimal schedule with finish times defined as $\mathcal{C}(R_\mathrm{a}, R_\mathrm{i})$.

Next, the last case to check is $c_i^* \geq c_{i+1}$. By the third condition of lemma 4, there are i jobs with release dates smaller or equal than $c_i - p$. Therefore, there must be at least one job j in ς^* with release date smaller or equal to $c_i - p$ and start time bigger than c_i^*. Since $c_i^* - p \geq c_i$ job, j' can be started at time c_i such that it finishes before c_i^*. Again, since the objective function is regular, this cannot increase the objective value. Therefore, if there is an optimal schedule with the first $i-1$ times equal to the first $i-1$ times of $\mathcal{C}(R_\mathrm{a}, R_\mathrm{i})$ there is also an optimal schedule with the first i times being equal. Therefore, there must be an optimal schedule among all feasible Π' schedules such that the first i finish times are equal to the first i times in $\mathcal{C}(R_\mathrm{a}, R_\mathrm{i})$. This is a contradiction to the assumption above.

The optimal schedule among these schedules can be found by solving an assignment problem in $O(n^3)$ as shown in Theorem 2. Since any feasible schedule in Π is feasible in Π', the objective value of the optimal schedule can only be lower than the objective value of the optimal $(R_\mathrm{a}, R_\mathrm{i})$-schedule in Π. Therefore, objective value of the solution is a lower bound for the objective value of the optimal $(R_\mathrm{a}, R_\mathrm{i})$-schedule. □

Appendix G Proof of Theorem 6

Proof The first step can be done in $O(n)$ after sorting the times in R_a. In the second step, we iterate over at most n^2 times in $\mathcal{S}$. For each $s \in \mathcal{S}$, we iterate over at most $2n$ entries in $R_\mathrm{a} \cup R(\mathcal{J})$ and n^2 entries in R_i such that the step finishes in $O(n^4)$.

In the third step for each job J_j, the set $T_\mathrm{act}^\mathrm{max}$ can be computed by iterating once over all times t in $R_\mathrm{a} \cup R(\mathcal{J})$ with at most $2n$ elements. For each t, one must iterate over R_i with at most n^2 times. To find S_j^min, one must iterate once over $T_\mathrm{act}^\mathrm{max}$ with at most n elements. Therefore, also the third step terminates in $O(n^4)$.

Finally the assignment problem can be solved in $O(n^3)$. □

Appendix H Proof of Theorem 7

Proof Suppose there is a job J_j such that there is no time $t \in R(\mathcal{J})$ with $t > \mathrm{Prev}(r_j, R_\mathrm{a}), t < \hat{t}$ and $t \notin R_\mathrm{i}$. Furthermore, assume that $\hat{t} \in R_\mathrm{a} \cup R_\mathrm{i}$ and that job J_j is started before its release date in $\varsigma_l(R_\mathrm{a}, R_\mathrm{i})$. We first show that $S_j^\mathrm{min} \geq \hat{t}$. Assume $S_j^\mathrm{min} < \hat{t}$. Then, by definition of S_j^min there must be a time $t \in T_\mathrm{act}^\mathrm{max}$ with $t \leq S_j^\mathrm{min} < \hat{t}$. By the conditions of the theorem, all release dates between $\mathrm{Prev}(r_j, R_\mathrm{a})$ and $\hat{t}$ are in R_i. Therefore, by the definition of $T_\mathrm{act}^\mathrm{max}$ it must hold $\mathrm{Prev}(r_j, R_\mathrm{a}) \in T_\mathrm{act}^\mathrm{max}$.

First assume $\hat{t} \notin R_\mathrm{a}$ such that $\hat{t} \in R_\mathrm{i}$ by definition of $\hat{t}$. Additionally, we know that $\hat{t} \geq \mathrm{Prev}(r_j, R_\mathrm{a}), \hat{t} \leq (r_j + (\mathrm{Prev}(r_j, R_\mathrm{a}) - r_j) \mod p)$ and $(\hat{t} - \mathrm{Prev}(r_j, R_\mathrm{a})) \mod p = 0$. Therefore, $\mathrm{Prev}(r_j, R_\mathrm{a})$ cannot be in $T_\mathrm{act}^\mathrm{max}$ which is a contradiction.

Now assume $\hat{t} \in R_\mathrm{a}$. Since additionally $\hat{t} \geq r_j$, it holds $\hat{t} \geq \mathrm{Next}(r_j, R_\mathrm{a})$. By definition of $\hat{t}$, it holds $\hat{t} = r_j + ((\mathrm{Prev}(r_j, R_\mathrm{a}) - r_j) \mod p)$. Since $\mathrm{Prev}(r_j, R_\mathrm{a})$ is the only time in $T_\mathrm{act}^\mathrm{max}$ smaller than $\hat{t}$, we can conclude that $S_j^\mathrm{min} = \hat{t}$ by the definition of S_j^min.

Since the assignment problem only assigns job J_j to start time s in case, it holds $r'_j \leq s$, J_j starts before its release date and we know $\mathrm{Next}(r_j + p, \mathcal{C}(R_\mathrm{a}, R_\mathrm{i})) - p > r_j$, we can conclude $S_j^\mathrm{min} < \mathrm{Next}(r_j + p, \mathcal{C}(R_\mathrm{a}, R_\mathrm{i})) - p$ and $|R_\mathrm{a}| < n$. First assume $\mathrm{Next}(r_j + p, \mathcal{C}(R_\mathrm{a}, R_\mathrm{i})) = \infty$. Then, $\mathrm{Prev}(r_j + p, \mathcal{C}(R_\mathrm{a}, R_\mathrm{i})) = \max(\mathcal{C}(R_\mathrm{a}, R_\mathrm{i}))$. By construction of $\mathcal{C}(R_\mathrm{a}, R_\mathrm{i})$, we know that $J(c_n - p) \geq n$ such that all n jobs are released at $\max \mathcal{C}(R_\mathrm{a}, R_\mathrm{i})$. Therefore, job J_j is not started before its own release date since $r'_j \geq r_j$.

Next assume $\mathrm{Next}(r_j + p, \mathcal{C}(R_\mathrm{a}, R_\mathrm{i})) = c_1$. By construction of c_1, it must hold $c_1 - p \leq \min(R_\mathrm{a})$, and therefore,

$$S_j^\mathrm{min} < \mathrm{Next}(r_j + p, \mathcal{C}(R_\mathrm{a}, R_\mathrm{i})) - p = c_1 - p \leq \min(R_\mathrm{a}).$$

Since $\mathrm{Next}(r_j, R_\mathrm{a}) \geq \min(R_\mathrm{a})$, we have the contradiction

$$S_j^\mathrm{min} < \min(R_\mathrm{a}) \leq \mathrm{Next}(r_j, R_\mathrm{a}) \leq S_j^\mathrm{min}.$$

Now assume $\mathrm{Next}(r_j + p, \mathcal{C}(R_\mathrm{a}, R_\mathrm{i})) = c_i$ with $i > 1$. By Theorem 5, it must hold $c_i - p \notin R_\mathrm{i}$. We first consider the case $c_i - p \in R_\mathrm{a}$. Since $\mathrm{Next}(r_j, R_\mathrm{a}) \in R_\mathrm{a}$, it holds $\mathrm{Next}(r_j, R_\mathrm{a}) + p \in \mathcal{C}(R_\mathrm{a}, R_\mathrm{i})$ with Theorem 5. Therefore,

we know

$$\text{Next}(r_j + p, \mathcal{C}(R_\text{a}, R_\text{i})) \leq \text{Next}(r_j, R_\text{a}) + p,$$

and subsequently,

$$\text{Next}(r_j + p, \mathcal{C}(R_\text{a}, R_\text{i})) - p \leq \text{Next}(r_j, R_\text{a}).$$

We have the contradiction

$$S_j^{\min} < \text{Next}(r_j + p, \mathcal{C}(R_\text{a}, R_\text{i})) - p \leq \text{Next}(r_j, R_\text{a}) \leq S_j^{\min}.$$

The last case to check is $c_i - p \notin R_\text{a}$. Since J_j is started after its release date in $\varsigma_l(R_\text{a}, R_\text{i})$, it must hold $c_i > r_j + p$.
Assume $\text{Next}(r_j, R_\text{a}) < c_i - p$. Then there must be a $t' \in R_\text{a}$ with $t' = \text{Next}(r_j, R_\text{a}) < c_i - p$ and $t' + p \in \mathcal{C}(R_\text{a}, R_\text{i})$ due to Theorem 5. This is a contradiction since $r_j + p \leq t' + p < c_i$, and therefore,

$$\begin{aligned}\text{Next}(r_j + p, \mathcal{C}(R_\text{a}, R_\text{i})) &\leq t' + p < c_i \\ &= \text{Next}(r_j + p, \mathcal{C}(R_\text{a}, R_\text{i})).\end{aligned}$$

Furthermore, we know that $c_i - p \notin R_\text{a}$, and therefore, it must hold $c_i - p < \text{Next}(r_j, R_\text{a})$. We have the contradiction

$$\begin{aligned}S_j^{\min} &< \text{Next}(r_j + p, \mathcal{C}(R_\text{a}, R_\text{i})) - p = c_i - p \\ &< \text{Next}(r_j, R_\text{a}) \leq S_j^{\min}.\end{aligned}$$

Therefore, the claim must be true and job J_j is not started before its own release date in $\varsigma_l(R_\text{a}, R_\text{i})$. □

Appendix I Proof of Theorem 8

Proof We use a similar approach as we did in computing the lower bound for the branch-and-bound algorithm. We denote the original instance as Π and an adapted instance as Π_u. We show that the single assignment heuristic solves Π_u optimally. We then define another instance Π_l with an optimal value that is guaranteed to be below or equal to the objective value of the optimal schedule of the original instance Π. Finally, we show that the difference of the optimal values of the two adapted problems is bounded by the term (6).
Let Π be an instance of $\text{MDS} - \text{EP}$. As stated before, we can assume that there is a feasible schedule with start times defined as $S = \{0, p, 2p, \ldots, (n-1)p\}$. Let Π_u be the same as Π, but with adapted release dates

$$r_j{}^{(u)} = \text{Next}(r_j, S).$$

Similar to previous results, we show that there must be an optimal schedule for Π_u with the start times given by S. The reasoning follows the same steps as before. There must be an optimal schedule such that the first job starts at 0. Assume otherwise an let s_1' be the first start time of any feasible schedule ς'. If $s' < p$, the release date of this first job must be 0 and the job could also be started at time 0 which is a contradiction. If $s' \geq p$, no job is processed between 0 and p. Since there is at least one job with release date 0, this job could be scheduled at time 0. In both cases, the new schedule is optimal as well, since all jobs are started at the same time or earlier and the objective function is regular.
Now assume there is an optimal schedule with the first i start times being equal to the first i times in S. Then there must also be an optimal schedule with the first $i + 1$ start times being equal. This can be shown in the same way as before. Either the job at position $i + 1$ can be started at c_i or there is enough time to schedule another job that is released before c_i but started at a later time. Therefore, there is also an optimal schedule with the first $i + 1$ start times being equal to the first $i + 1$ start times in S. Since it also holds for $i = 1$, there must be an optimal schedule with the start times defined by S.
The assumption that there is a feasible schedule with start times

$$S = \{0, p, 2p, \cdots, (n-1)p\}$$

ensures that the set of finish times C constructed of Eq. 5 is equal to

$$\{p, 2p, \cdots, np\}.$$

Since the single assignment problem finds the optimal schedule among all schedules with these times, the single assignment heuristic solves Π_u optimally (Fig. 5).
We define Π_l by reducing some of the original release dates. Since all feasible schedules of the original instance Π are feasible in Π_l as well, the objective value of the optimal schedule may only be improved. We set

$$r_j{}^{(l)} = \text{Prev}(r_j, S).$$

Let ς_l^* be an optimal schedule of Π_l and ς_u^* an optimal schedule of Π_u. We know that for the optimal schedule ς^* of the original problem Π it holds

$$W(\varsigma_l^*) \leq W(\varsigma^*) \leq W(\varsigma_u^*).$$

Furthermore, we know that for any feasible schedules ς_u of Π_u and ς_l of Π_l it holds $W(\varsigma_u^*) \leq W(\varsigma_u)$ and $W(\varsigma_l^*) \leq W(\varsigma_l)$. In the following, we construct a feasible schedule ς_u for Π_u and show that it can be transformed into ς_l^* by increasing the objective value by at most the bound of the optimality gap given in the Lemma. To do so, assume the optimal schedule of Π_l is ς_l^*. Next we construct a feasible

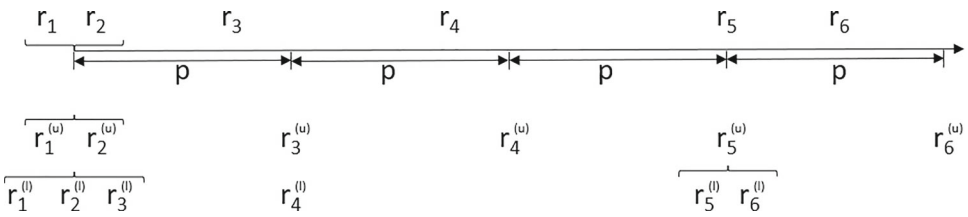


Fig. 5 Construction of $r_j^{(u)}$ and $r_j^{(l)}$

Table 2 Example for Algorithm 2

ς_l^*	J_4	J_3	J_1	J_2	J_6	J_5
ς_u	J_1	J_4	J_3	J_2	J_5	J_6
ς_l^1	J_1	J_4	J_3	J_2	J_5	J_6
ς_l^2	J_4	J_1	J_3	J_2	J_5	J_6
ς_l^3	J_4	J_3	J_1	J_2	J_5	J_6
ς_l^4	J_4	J_3	J_1	J_2	J_5	J_6
ς_l^5	J_4	J_3	J_1	J_2	J_5	J_6
ς_l^6	J_4	J_3	J_1	J_2	J_6	J_5

schedule ς_u. We assign the job with the lowest release date to the first position. If multiple such jobs exists, we take an arbitrary job. At position $i+1$, we start the job that is in position i in ς_l^*. We repeat this process until we reach the position of the first job in ς_l^*. Then, we repeat this process with the job with the lowest release among all jobs that have not been assigned yet. Again we solve any tie arbitrarily. Algorithm 2 gives a formal definition for this procedure. For easier readability, we write $\varsigma_u(k)$ for the job in the kth position of the schedule ς_u. □

Algorithm 2 ***Construction of*** ς_u

1. *Set* $i = 1$.
2. *Let* J_j *be the job with the smallest release date among all jobs that have not yet been assigned to a start time. Let* l *be the position of* J_j *in* ς_l^*.
3. *If* l *is equal to* i*, set* $i = i+1$ *and go to the previous step. For* $k = i+1, \ldots, l$*, set* $\varsigma_u(k) = \varsigma_l^*(k-1)$.
4. *If* $k = n$*, terminate the algorithm. Otherwise set* $i = k+1$ *and go to step 2.*

Proof Table 2 shows an example for a schedule ς_u constructed from ς_l^*. Additionally, the schedules ς_l^i used later are also depicted. Recall that we assume $r_i \leq r_{i+1}$.

Note that by construction of ς_u each job is assigned exactly once. Furthermore, for each job J_j it holds true that it is either assigned to position $k > j$ or it is exactly one position behind its position in ς_l^*. Indeed, if the kth job is assigned in step 2 of the previous algorithm, there are at most $k-1$ jobs with lower release date. If the kth job is assigned in step 3, it is one position behind its position in ς_l^*. By assumption, scheduling a job with the kth lowest release date or a smaller release date in the kth position must be feasible for Π_u. Furthermore, it holds by construction $r_j{}^{(u)} = r_j{}^{(l)}$ if $r_j = r_j^u$ or $r_j^u = r_j^l + p$. Therefore, starting job J_j p time units after its start in ς_l^* must be feasible for Π_u. Therefore, the whole schedule is feasible for Π_u.

Finally, we bound the difference between ς_l^* and ς_u. Set $\varsigma_l^1 = \varsigma_u$. For $i \in \{2, \ldots, n\}$, define $\varsigma_l^i = \varsigma_l^{i-1}$, but change the position of the i-th job with the $(i-1)$-th job in case $\varsigma_u(i-1) \neq \varsigma_l{}^*(i-1)$. By construction, the first $i-1$ jobs of ς_l^i are the same as in ς_l^*. Since the jobs in position $1, \ldots, n-1$ are the same for ς_l^n, also the job in position n must be the same. Therefore, by construction it holds $W(\varsigma_u) = W(\varsigma_l^1)$ and $W(\varsigma_l^n) = W(\varsigma_l^*)$. Furthermore, since only the jobs in position i+1 and i may be switched it holds

$$W(\varsigma_l^i) - W(\varsigma_l^{i+1}) \leq \Delta_w^{\max}(i) - \Delta_w^{\min}(i).$$

with $\Delta_w^{\max}(i)$ and $\Delta_w^{\min}(i)$ as defined in the Theorem. We now have the necessary results to conclude

$$\begin{aligned} &W(\varsigma_u^*) - W(\varsigma^*) \leq W(\varsigma_u^*) - W(\varsigma_l^*) \\ &\leq W(\varsigma_u) - W(\varsigma_l^*) \\ &\leq W(\varsigma_u) - W(\varsigma_l^1) + W(\varsigma_l^1) - W(\varsigma_l^2) \\ &\quad + \cdots + W(\varsigma_l^n) - W(\varsigma_l^*) \\ &\leq \sum_{k=1}^{n-1} \Delta_w^{\max}(k) - \sum_{k=1}^{n-1} \Delta_w^{\min}(k). \end{aligned}$$

□

Appendix J Proof of Corollary 5

Proof Denote by I_k the interval $\left[kp, (k+1)p - 1\right]$ for $0 \leq k \leq n-1$. Since the minimum distance between two due dates for a single job is at least p, there is at most one due date per job in each interval I_k. By definition, it holds

$$w_j((k+1)p) - w_j(kp) = \begin{cases} w_{j,l} & \text{if } \exists D_{j,l} \in I_k \\ 0 & \text{else.} \end{cases}$$

Due to the minimum time distance $t^{\min}$ between two due dates, there are at most $\lceil \frac{np}{t^{\min}} \rceil$ intervals with due dates. Since it holds for all jobs $w_j((k+1)p) - w_j(kp) = 0$ for intervals

with no due dates, we can follow

$$\sum_{k=1}^{n-1} \Delta_w^{\max}(k) - \sum_{k=1}^{n-1} \Delta_w^{\min}(k) \leq \sum_{k=1}^{n-1} \Delta_w^{\max}(k) \leq w_j^{\max} \left\lceil \frac{np}{t^{\min}} \right\rceil.$$

□

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Baptiste, P. (1999). Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *Journal of Scheduling, 2*(6), 245–252. https://doi.org/10.1002/(SICI)1099-1425(199911/12)2:6<245::AID-JOS28>3.0.CO;2-5

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

Brucker, P. (2007). *Scheduling Algorithms, 5* (ed). Springer-Verlag GmbH.

Cheng, T. C. E., & Gupta, M. C. (1989). Survey of scheduling research involving due date determination decisions. *European Journal of Operational Research, 38*(2), 156–166. https://doi.org/10.1016/0377-2217(89)90100-8

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*. S.l.: A series of books in the mathematical sciences. W H Freeman.

Gordon, V., & Kubiak, W. (1998). Single machine scheduling with release and due date assignment to minimize the weighted number of late jobs. *Information Processing Letters, 68*(3), 153–159. https://doi.org/10.1016/S0020-0190(98)00153-7

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 5*, 287–326. https://doi.org/10.1016/S0167-5060(08)70356-X

Gafarov, E. R., Lazarev, A. A., & Werner, F. (2020). Minimizing total weighted tardiness for scheduling equal-length jobs on a single machine. *Automation and Remote Control, 81*(5), 853–868. https://doi.org/10.1134/S0005117920050069

Gordon, V., Proth, J.-M., & Chu, C. (2002). A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research, 139*(1), 1–25. https://doi.org/10.1016/S0377-2217(01)00181-3

Hariri, A. M. A., & Potts, C. N. (1994). Single machine scheduling with deadlines to minimize the weighted number of tardy jobs. *Management Science, 40*(12), 1712–1719. https://doi.org/10.1287/mnsc.40.12.1712

Hertrich, C., Weiß, C., Ackermann, H., Heydrich, S., & Krumke, S. O. (2020). Scheduling a proportionate flow shop of batching machines. *Journal of Scheduling, 23*(5), 575–593. https://doi.org/10.1007/s10951-020-00667-2

Hertrich, C., Weiß, C., Ackermann, H., Heydrich, S., & Krumke, S. O. (2022). Online algorithms to schedule a proportionate flexible flow shop of batching machines. *Journal of Scheduling, 25*(6), 643–657. https://doi.org/10.1007/s10951-022-00732-y

Jackson, J.R.: Scheduling a Production Line to Minimize Maximum Tardiness. Research Report 43, Mgmt. Sci. Research Project, UCLA, Los Angeles (1955)

Lawler, E. L. (1977). A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics, 1*, 331–342. https://doi.org/10.1016/S0167-5060(08)70742-8

Lawler, E.L.: Scheduling a single machine to minimize the number of late jobs. Technical Report UCB/CSD-83-139, EECS Department, University of California, Berkeley (Oct 1983). http://www2.eecs.berkeley.edu/Pubs/TechRpts/1983/6344.html

Lawler, E. L., & Moore, J. M. (1969). A functional equation and its application to resource allocation and sequencing problems. *Management Science, 16*(1), 77–84. https://doi.org/10.1287/mnsc.16.1.77

Lenstra, J. K., Rinnooy Kan, A. H. G., & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics, 1*, 343–362. https://doi.org/10.1016/S0167-5060(08)70743-X

Moore, J. M. (1968). An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science, 15*(1), 102–109. https://doi.org/10.1287/mnsc.15.1.102

Shabtay, D. (2016). Optimal restricted due date assignment in scheduling. *European Journal of Operational Research, 252*(1), 79–89. https://doi.org/10.1016/j.ejor.2015.12.043

Shabtay, D., & Steiner, G. (2006). Two due date assignment problems in scheduling a single machine. *Operations Research Letters, 34*(6), 683–691. https://doi.org/10.1016/j.orl.2005.10.009

van den Akker, J. M., Diepen, G., & Hoogeveen, J. A. (2010). Minimizing total weighted tardiness on a single machine with release dates and equal-length jobs. *Journal of Scheduling, 13*(6), 561–576. https://doi.org/10.1007/s10951-010-0181-1