Single machine rescheduling for new orders with maximum and total time disruption constraints

Journal of Scheduling
https://doi.org/10.1007/s10951-025-00837-0

Single machine rescheduling for new orders with maximum and total time disruption constraints

Elena Rener[1,3] · **Fabio Salassa**[1] · **Vincent T'kindt**[1,2]

Accepted: 12 January 2025

Abstract
Rescheduling problems arise in a variety of situations where a previously planned schedule needs to be adjusted to deal with unforeseen events. A common problem is the arrival of new orders, i.e., jobs, which have to be integrated into the schedule of the so-called old jobs. The maximum and total absolute time deviations of the completion times of these jobs are modeled as a disruption constraint to limit the change in the original schedule. Disruption constraints influence the shape of an optimal schedule in particular with respect to the sequencing of old jobs and the insertion of idle time. In this paper, a classification into idle and no-idle problems is given, through several examples, for a set of single machine rescheduling problems with different objective functions. We also prove the complexity of four rescheduling problems that have been left open in the literature. Finally, algorithms are provided to solve the timing problem that arises for certain single machine rescheduling problems when the order of the jobs is fixed.

Keywords Rescheduling · Single machine · New orders · Maximum disruption · Total disruption

1 Introduction

In the literature, rescheduling has been considered to be a way of modifying an existing schedule to respond to unforeseen disruptive events, such as the arrival of new jobs, the delay of some other jobs, machine breakdowns, or periods of unavailability of machines due to maintenance activities. The aim of the present work is to study scheduling strategies when unexpected jobs, called *new jobs*, have to be scheduled in an already given optimal solution made up of known jobs, called *old jobs*. This is typically called rescheduling for new orders. These strategies seek a trade-off between limiting the perturbation of the original schedule, called *disruption* of the schedule, and integrating the new jobs to optimize an objective function.

Rescheduling problems for new orders have been the subject of a number of studies. This family of rescheduling problems was first formalized by Hall and Potts (2004) and has since received much attention. Hall and Potts (2004) studied several single machine scheduling problems. They considered total completion time or maximum lateness as objective functions to be minimized in combination with a disruption cost. They provided polynomial-time algorithms or showed $\mathcal{NP}$-hardness for each of the problems tackled. The same problems were studied by Teghem and Tuyttens (2014), who considered the enumeration of Pareto optima. Zhao et al. (2016) introduced a generalization of the disruption criterion by imposing on each old job j a maximum amount of allowed disruption k_j. In this case, both the problems of minimizing the sum of the completion times and the maximum lateness become $\mathcal{NP}$-hard. Only two research papers were found that consider rescheduling problems with multiple disruptions (Hall et al., 2007; Yuan et al., 2007). The setting here changes in the sense that there are multiple sets of new jobs arriving at different times. Thus, the initial state does not necessarily imply optimal schedules. In addition to

✉ Elena Rener
elena.rener@polito.it

Fabio Salassa
fabio.salassa@polito.it

Vincent T'kindt
tkindt@univ-tours.fr

[1] Dipartimento di Ingegneria Gestionale e della Produzione, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy

[2] Laboratoire d'Informatique Fondamentale et Appliquée, Université de Tours, 64 Avenue Jean Portalis, 37200 Tours, France

[3] Department of Business Administration, University of Bern, Engehaldenstrasse 4, 3012 Bern, Switzerland

Published online: 13 March 2025

multiple disruptions, Yuan et al. (2007) also considered jobs with release dates. Yuan and Mu (2007) studied four single machine rescheduling problems with minimization of the makespan in the presence of job release dates and different disruption criteria. Complexity results were given for the different scenarios. Yang (2007), Zhao and Tang (2010), and Liu and Zhou (2015) considered the single machine rescheduling problem when jobs have time-dependent processing times. Guo et al. (2021) studied a rescheduling problem motivated by energy savings. New orders are rework jobs that need to be added to the original schedule to minimize the total waiting time of the jobs. Liu et al. (2018) investigated the rescheduling of a two-machine flow shop with outsourcing solved by means of a hybrid variable neighborhood search. Zhang et al. (2022) took into consideration the minimization of the maximum weighted tardiness for rescheduling problems. Rener et al. (2022) studied the rescheduling problem for minimizing the maximum lateness with a constraint on the total time disruption and proposed the first exact algorithm to solve the problem when a no-idle constraint is required. Fang et al. (2023) studied the generalization of the problem with rejection for minimizing a linear combination of the total weighted completion time, the maximum disruption, and the rejection cost. They first proved the $\mathcal{NP}$-hardness of the problem and then solved it with an exact dynamic programming algorithm and a fully polynomial-time approximation algorithm. They showed the efficiency of the proposed algorithms by means of computational experiments. A more efficient dynamic programming algorithm for the problem without rejection is proposed by Lendl et al. (2024). In addition, the authors present two fully polynomial-time approximation schemes and a constant-bound approximation polynomial algorithm for the same problem. Further complexity results are provided for related rescheduling problems.

We refer the readers to the surveys by Vieira et al. (2003) and Aytug et al. (2005) for a general introduction to rescheduling problems. More recent papers covering the topic from different perspectives are proposed for instance by Pferschy et al. (2023), Luo et al. (2022), Wang et al. (2018).

In this paper, we consider rescheduling problems for new orders arising from the combination of two disruption criteria and several scheduling objective functions. Our contributions, which focus on single machine problems, where all jobs must be scheduled and no rejection is allowed, can be summarized as follows:

- we provide structural properties, when the objective is to minimize classical scheduling objective functions, subject to a constraint on the maximum or total absolute time disruption of the original jobs;
- we propose a classification of rescheduling problems according to whether machine idle times may be required in optimal solutions;
- we provide proofs of the computational complexity of four open problems;
- we propose two novel polynomial-time timing algorithms for solving the problems that require idle times to be inserted for the case where a fixed sequence of jobs is given.

The remainder of the paper is organized as follows: in Sect. 2, we formally define rescheduling problems and give contributions regarding the structure of optimal solutions. We present both properties for preserving the order of old jobs after rescheduling and properties on the requirement of inserting idle times. In Sect. 3, we define the computational complexity of some open problems. In Sect. 4, we study three sets of rescheduling problems, when the order of jobs is given, and propose polynomial-time algorithms for solving the respective timing problems. Finally, Sect. 5 gives the conclusions and directions for future work.

2 Structural properties

2.1 Notation

Let J^O (resp. J^N) be the set of n_O old jobs (resp. n_N new jobs) that must be processed on a single machine. Each job j has a processing time p_j and, depending on the problem, may have a due date d_j or a weight w_j. Given any schedule σ, let $C_j(\sigma)$ (C_j when there is no ambiguity) be the completion time of job j. Initially, jobs in J^O are assumed to be optimally scheduled in a schedule π^* to minimize a given objective function f. After the arrival of jobs from J^N, the goal is to generate a new schedule σ^* of all jobs to minimize f while not disrupting π^* too much. According to the seminal work by Hall and Potts (2004) that introduced this class of problems, the disruption of the schedule is modeled as the maximum or the sum of the absolute deviations of the completion times of old jobs. Let $\Delta_{max} = max_{j\in J^O}(\Delta_j)$ be the maximum disruption and $\overline{\Delta} = \sum_{j\in J^O} \Delta_j$ be the total disruption, where the disruption is given by $\Delta_j = |C_j(\sigma^*) - C_j(\pi^*)|, \forall j \in J^O$. Notice, that since we consider absolute time deviations of completion times, both having a job early and late w.r.t. the original completion time causes a disruption. This may be encountered in practice whenever there are other dependencies at an operational level that rely on the original completion times of jobs (e.g., material supply, delivery times, ...). A threshold ϵ limits the amount of the schedule disruption (the total or the maximum disruption depending on the problem at hand). Each job in a schedule is associated with a function f_j, which we assume to be regular, i.e., increasing with respect to the job completion time. We consider, for each problem, an objective

function $f \in \{f_{max}, \overline{f}\}$, where $f_{max} = \max_{j \in J^O \cup J^N}(f_j)$ and $\overline{f} = \sum_{j \in J^O \cup J^N} f_j$. Whenever interesting, we will consider the following particular cases of $\overline{f}$ and f_{max}. Among the $\overline{f}$ objectives, we consider three different objective functions:

- the total weighted completion time $\overline{wC}$, where for each job $f_j = w_j C_j$;
- the total number of late jobs $\overline{U}$, where for each job $f_j = 1$ if $C_j > d_j$ and $f_j = 0$ otherwise;
- the total tardiness $\overline{T}$, where for each job $f_j = \max(0, C_j - d_j)$.

As a f_{max} objective, we consider the most widely used in the scheduling literature, which is the maximum lateness L_{max}, where $f_j = C_j - d_j$.

Using the classical three-field notation of Graham et al. (1979), the problems we consider in this work are rescheduling problems on a single machine in the form of $1|\Delta_{max} \leq \epsilon|f$ and $1|\overline{\Delta} \leq \epsilon|f$.

2.2 Preserving the initial order for old jobs

Old jobs are initially sequenced in a schedule π^* to minimize an objective function f. This order might be preserved in an optimal schedule of the rescheduling problem. In this section, we focus on establishing some properties of when this order is preserved, as opposed to when a *re-sequencing* of old jobs is required.

The first two properties illustrated below serve a dual purpose. The first is to provide the source of the need for idle time insertion. The second is to provide actual structural properties for the related problems $1|f \leq \epsilon|\Delta_{max}$ and $1|f \leq \epsilon|\overline{\Delta}$. The two disruption criteria lead to different properties. We first show that problem $1|f \leq \epsilon|\Delta_{max}$ never requires a re-sequencing to obtain an optimal solution, if any feasible schedule exists with old jobs ordered as in π^*. On the contrary, we show that problem $1|f \leq \epsilon|\overline{\Delta}$ may require a re-sequencing of old jobs, even if there exists a feasible schedule with old jobs ordered as in π^*.

Property 1 *If there exists a non-empty set of feasible schedules for problem $1|f \leq \epsilon|\Delta_{max}$ with old jobs ordered as in π^*, there exists an optimal solution that belongs to this set of schedules.*

Proof Consider an initial schedule π^* made of old jobs, and two schedules σ' and σ^* made of old and new jobs, as illustrated in Fig. 1. Schedule π^* shows the original optimal schedule with old jobs only. Schedule σ' is a schedule with old and new jobs, where at least two old jobs i and j are swapped with respect to their ordering in π^*. Suppose that σ' is an optimal solution for the problem $1|f \leq \epsilon|\Delta_{max}$.

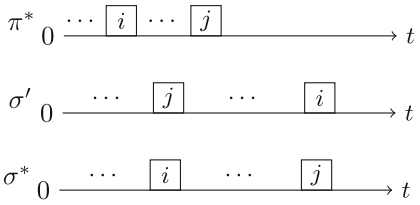


Fig. 1 Jobs sequencing for the $1|f \leq \epsilon|\Delta_{max}$ problem

Now, consider any old job and its successors in π^*. Let i be the first old job that follows, in σ', at least one of its successors in π^*, and let j be the last old job scheduled before i in σ'. Notice that in σ', before job j there can be both old and new jobs, while between j and i there can be only new jobs because j is the closest old job that precedes i in σ'. Let σ^* be the same schedule than σ' but with i and j swapped, such that the starting time of j in σ' is the same of i in σ^* and $C_i(\sigma') = C_j(\sigma^*)$. In σ^*, before job i there can be both old and new jobs, while between i and j there can be new jobs only. Since σ' is assumed to be optimal, $\Delta_{max}(\sigma') \leq \Delta_{max}(\sigma^*)$.

The proof is done by contradiction showing that, if there exists such a schedule σ^*, with $f \leq \epsilon$, then $\Delta_{max}(\sigma^*) < \Delta_{max}(\sigma')$ and σ' cannot be optimal.

First, notice that $C_i(\sigma^*) - C_i(\pi^*) > 0$ by definition of job i and that $\Delta_i(\sigma^*) < \Delta_i(\sigma')$ since $C_i(\sigma') > C_i(\sigma^*)$.

Given that $C_j(\pi^*) > C_i(\pi^*)$ and $C_i(\sigma') = C_j(\sigma^*)$, we have that $\Delta_j(\sigma^*) < \Delta_i(\sigma')$.

Next, we show that $\Delta_j(\sigma') \leq \Delta_i(\sigma')$. If $C_j(\sigma') \geq C_j(\pi^*)$, then this holds, since i is more right-shifted than j. If $C_j(\sigma') < C_j(\pi^*)$, we have $\Delta_j(\sigma') \leq |C_j(\sigma') - C_j(\sigma^*)|$. For the disruption of job i it holds $\Delta_i(\sigma') \geq C_i(\sigma') - C_i(\sigma^*)$ and since $|C_j(\sigma') - C_j(\sigma^*)| = C_i(\sigma') - C_i(\sigma^*)$, we have $\Delta_j(\sigma') \leq |C_j(\sigma') - C_j(\sigma^*)| = C_i(\sigma') - C_i(\sigma^*) \leq \Delta_i(\sigma')$.

Putting all together, we obtain

$$\max(\Delta_i(\sigma'), \Delta_j(\sigma')) = \Delta_i(\sigma') > \max(\Delta_i(\sigma^*), \Delta_j(\sigma^*)),$$

and since all other old jobs have the same starting and completion times in σ' and σ^*,

$$\Delta_{max}(\sigma') > \Delta_{max}(\sigma^*).$$

Repeating the argument for any pair of old jobs i and j defined as above proves the statement.

Now consider the problem $1|f \leq \epsilon|\overline{\Delta}$. The following property holds.

Property 2 *For problems $1|f \leq \epsilon|\overline{\Delta}$, an optimal schedule may have old jobs scheduled in a different order with respect to π^* even if swapping them keeps the schedule feasible with respect to the constraint $f \leq \epsilon$.*

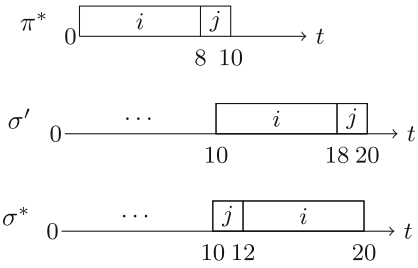


Fig. 2 Jobs sequencing in the $1|f \leq \epsilon|\overline{\Delta}$ problem

Proof Consider a schedule with two old jobs i and j with $p_i = 8$ and $p_j = 2$ scheduled consecutively so that $C_i(\pi^*) = 8$ and $C_j(\pi^*) = 10$ (see Fig. 2).

Schedule σ' is a schedule with old and new jobs, where the two old jobs are scheduled as in π^*. Suppose that σ' is optimal for solving $1|f \leq \epsilon|\overline{\Delta}$. Let σ^* be a feasible schedule, with all jobs, except jobs i and j, which are swapped, scheduled as in σ'. Assume that the starting time of job i in σ' as well as the starting time of job j in σ^* is $t = 10$. Then, the optimal sequencing of the jobs is (j, i) with a resulting $\overline{\Delta}(\sigma^*) = 14 < \overline{\Delta}(\sigma') = 20$. Since any other new job scheduled does not have an effect on the total disruption, schedule σ' is suboptimal.

2.3 Relevance of idle time insertion

Usually, minimizing a regular objective function allows us to consider only the sequencing or permutation problem in scheduling. However, for our rescheduling problems, the constraint on the total disruption implicitly models a second objective that is nonregular. So, inserting idle times becomes relevant for solving the problem. Several considerations and examples are given to provide insights into when the insertion of machine idle times is required.

Consider the following example. Three old jobs are originally scheduled in $\pi^* = (i, j, k)$. Let be $p_i = 6, p_j = p_k = 1$. Assume that the minimum cost schedule with no inserted idle times, that includes a new job h, with $p_h = 1$, is $\sigma^*_{no-idle} = (h, j, k, i)$ as in Fig. 3. In schedule $\sigma^*_{no-idle}$, we have $C_j(\pi^*) - C_j(\sigma^*_{no-idle}) > C_i(\sigma^*_{no-idle}) - C_i(\pi^*)$ and $C_k(\pi^*) - C_k(\sigma^*_{no-idle}) > C_i(\sigma^*_{no-idle}) - C_i(\pi^*)$. In this case, constructing a schedule σ^*_{idle} with one unit of idle time before job j reduces both Δ_{max} and $\overline{\Delta}$. So, in this example, for the Δ_{max} criterion and $\epsilon = 4$, schedule σ^*_{idle} makes this schedule of jobs feasible, and possibly optimal, as well as for the $\overline{\Delta}$ criterion and $\epsilon = 12$.

The example gives a first feeling of the correlation between the re-sequencing of old jobs and the insertion of idle times to obtain optimal solutions. Notice that the decrease of the disruption by right shifting the jobs is due to the fact that jobs j and k were scheduled before their original completion time, and i after.

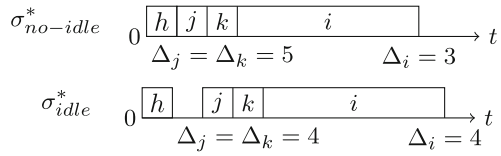


Fig. 3 Decrease of the schedule disruption via idle time insertion

The following property applies to an optimal schedule, where the old jobs are sequenced as in the original schedule.

Property 3 *If there exists an optimal solution in which old jobs are ordered as in the original schedule π^*, then this solution is without inserted idle times.*

Proof If all jobs in J^O are in the same order as in schedule π^*, the disruption of any job j in an optimal schedule σ^* is given by $\Delta_j = \sum_{i \in J^N : i \to j} p_i$, where $i \in J^N : i \to j$ denote the new jobs i that precede j in the schedule. By inserting an idle time $\delta > 0$ before job j, the disruption changes to $\Delta'_j = \sum_{i \in J^N : i \to j} p_i + \delta$. Then, in this new schedule σ', $\Delta_{max}(\sigma') \geq \Delta_{max}(\sigma^*)$ as well as $\overline{\Delta}(\sigma') > \overline{\Delta}(\sigma^*)$.

Also, $f(C_j + \delta) > f(C_j), \ \forall j \in J^O \cup J^N$ since the f_j's are regular functions. Then, $f_{max}(\sigma') \geq f_{max}(\sigma^*)$ as well as $\overline{f}(\sigma') > \overline{f}(\sigma^*)$.

Hence, inserting any idle time does not decrease the considered objective functions or the disruption criteria.

Let us now consider some rescheduling problems with particular objective functions. Hereafter, we consider the particular cases $f_{max} = L_{max}$ and $\overline{f} \in \{\overline{C}; \overline{wC}; \overline{U}; \overline{T}\}$.

Proposition 1 (Hall & Potts, 2004; Lendl et al., 2024) *There exists an optimal solution to problems $1|\Delta_{max} \leq \epsilon|f$, $f \in \{L_{max}, \overline{C}, \overline{wC}\}$, and $1|\overline{\Delta} \leq \epsilon|\overline{C}$, where the jobs are processed consecutively without inserted idle times.*

Hall and Potts (2004) provide the result for the problems with L_{max} and $\overline{C}$, while Lendl et al. (2024) consider the case with $\overline{wC}$. In all four cases, Property 3 is shown to hold and therefore, it is not necessary to consider idle time insertion to obtain optimal solutions.

Proposition 2 *An optimal solution to problems $1|\Delta_{max} \leq \epsilon|f_1$, $f_1 \in \{\overline{U}, \overline{T}\}$, and $1|\overline{\Delta} \leq \epsilon|f_2$, $f_2 \in \{L_{max}, \overline{wC}, \overline{U}, \overline{T}\}$ may contain inserted idle times.*

Proof To prove each of the stated results we provide, for each problem, a counterexample that shows the sub-optimality of a schedule with no idle time inserted.

We start with the problem $1|\Delta_{max} \leq \epsilon|\overline{U}$. Consider the schedule π^* in Fig. 4, with two old jobs i and j, with $p_i = 8$, $p_j = 1$ and $d_i = 8$, $d_j = 9$ and two new jobs k and h with $p_k = p_h = 1$ and due dates $d_k = d_h = 8$. Let be $\epsilon = 5$. An optimal solution for this instance is given by schedule

σ_{idle} with $\overline{U} = 1$, and where the idle time enables to meet the disruption constraint. In contrast, restricting the set of schedules to those without inserted idle times leads to the optimal solution $\sigma_{no-idle}$ with $\overline{U} = 2$.

Next, we consider the problem $1|\Delta_{max} \leq \epsilon|\overline{T}$. Consider jobs i, j, k, h from the above example and $\epsilon = 5$. Consider an additional old job ℓ with $p_\ell = 1$ and $d_\ell = 10$. Schedule π^* is shown in Fig. 5. The optimal schedule reaches $\overline{T} = 5$. In contrast, restricting the set of schedules to those without inserted idle times leads to the optimal solution $\sigma_{no-idle}$ with $\overline{T} = 7$.

Now, we turn to the problem $1|\overline{\Delta} \leq \epsilon|L_{max}$. Let be i, j, h the jobs in J^O with $p_i = 8$, $p_j = p_h = 1$ and $d_i = 12$, $d_j = 13$, $d_h = 14$ and k a new job with $p_k = 7$ and $d_k = 8$. Let us have $\epsilon = 10$. As illustrated in Fig. 6, we assume that $\pi^* = (i, j, h)$. the optimal schedule is $\sigma^*_{idle} = (k, j, h, i)$ with $L_{max} = 6$ with an idle time of 1 unit before job j. In contrast, restricting to the set of schedules without inserted idle times leads to the optimal solution $\sigma_{no-idle}$ with $L_{max} = 8$.

Next, we consider the problem $1|\overline{\Delta} \leq \epsilon|\overline{wC}$. Consider jobs $i, j, h \in J^O$ and $k \in J^N$ with $p_i = 5$, $p_j = p_h = 1$, $p_k = 4$ and $w_i = 7$, $w_j = w_h = 1$, $w_k = 12$. For $\epsilon = 7$, the optimal schedule is $\sigma^*_{idle} = (k, j, h, i)$ with $\overline{wC} = 145$ with an idle time of 1 unit before job j (Fig. 7). However, by imposing that no inserted idle time is allowed, the optimal solution becomes $\sigma^*_{no-idle}$ with $\overline{wC} = 172$.

Finally, we consider problems $1|\overline{\Delta} \leq \epsilon|\{\overline{U}, \overline{T}\}$. Set J^O is made up of the three jobs i, j, ℓ with $p_i = 8$, $p_j = p_\ell = 1$ and $d_i = 8$, $d_j = 9$, $d_\ell = 10$ while J^N is made by two new jobs k and h with $p_k = p_h = 1$ and $d_k = d_h = 8$. Let be $\epsilon = 15$ and the optimal solution with $\overline{U} = 1$ is given by schedule $\{k, h, j, \ell, i\}$ with one unit of idle time before j (Fig. 8). However, by imposing that no inserted idle time is allowed, the optimal solution becomes $\sigma_{no-idle}$ with $\overline{U} = 2$.

The same counterexample holds for $\overline{f}$ is considered to be $\overline{T}$, with a resulting objective $\overline{T} = 5$ of an optimal schedule with idle time and $\overline{T} = 7$ when imposing a constraint of no inserted idle time.

Table 1 summarizes the above results: for each couple of objective functions and disruption constraints, we indicate *mit* (*machine idle time*) when inserting machine idle times may be necessary to compute optimal solutions. We indicate *n-mit* (*no machine idle time*) for problems whose optimal solutions are without inserted idle times.

3 Complexity results

In this section, we recall complexity results of rescheduling problems from the literature and provide proofs for some open problems. Table 2 summarizes the current state of known complexity results. The first two rows contain the results stated by Hall and Potts (2004). Besides, very recently, Lendl et al. (2024) provide new results given by the following theorem.

Table 1 Problem classification w.r.t. the insertion of machine idle times

f	Δ_{max}	$\overline{\Delta}$
L_{max}	n-mit	mit
$\overline{C}$	n-mit	n-mit
$\overline{wC}$	n-mit	mit
$\overline{U}$	mit	mit
$\overline{T}$	mit	mit

Table 2 Computational complexity of rescheduling problems

f	Δ_{max}	$\overline{\Delta}$
L_{max}	$O(n + n_N log(n_N))$	strongly $\mathcal{NP}$-hard
$\overline{C}$	$O(n + n_N log(n_N))$	$\mathcal{NP}$-hard
$\overline{wC}$	$\mathcal{NP}$-hard	**$\mathcal{NP}$-hard**
$\overline{U}$	strongly $\mathcal{NP}$-hard	**strongly $\mathcal{NP}$-hard**
$\overline{T}$	**$\mathcal{NP}$-hard**	**strongly $\mathcal{NP}$-hard**

Theorem 1 (Lendl et al., 2024) *The following complexity results are established:*

1. *Problem* $1|\Delta_{max} \leq \epsilon|\overline{wC}$ *is weakly* $\mathcal{NP}$*-hard.*
2. *Problem* $1|\Delta_{max} \leq \epsilon|\overline{U}$ *is strongly* $\mathcal{NP}$*-hard.*

The complexity status in bold of Table 2 are proved in this section.

3.1 Minimizing the total number of tardy jobs/total tardiness with a constraint on the total disruption

Theorem 2 *Problems* $1|\overline{\Delta} \leq \epsilon|\overline{U}$ *and* $1|\overline{\Delta} \leq \epsilon|\overline{T}$ *are strongly* $\mathcal{NP}$*-hard.*

Proof Consider the 3-PARTITION problem, where a set S of elements $a_1, \ldots, a_{3t}$ is given. A solution to the problem exists if there exists an integer $B = \sum_j a_j/t$ such that the elements can be partitioned into sets $S_1, \ldots, S_t$, such that for every set S_j, $j = 1, \ldots, t$ $\sum_{i \in S_j} a_i = B$ and $|S_j| = 3$. The problem is known to be NP-complete in the strong sense. We show that the two rescheduling problems are strongly $\mathcal{NP}$-hard by reduction from 3-PARTITION, following the line of the proof in Hall and Potts (2004).

First, we show that 3-PARTITION reduces to the decision version of $1|\overline{\Delta} \leq \epsilon|\overline{U}$ denoted by $1|\overline{\Delta} \leq \epsilon, \overline{U} \leq 0|-$.

Consider the following instance of the rescheduling problems with $2t$ old jobs and $3t$ new jobs.

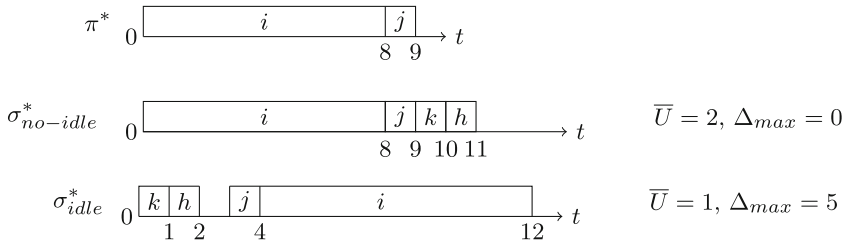


Fig. 4 Idle time insertion for the $1|\Delta_{max} \leq \epsilon|\overline{U}$ problem

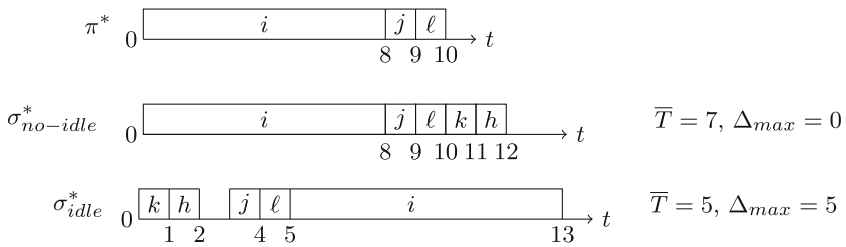


Fig. 5 Idle time insertion for the $1|\Delta_{max} \leq \epsilon|\overline{T}$ problem

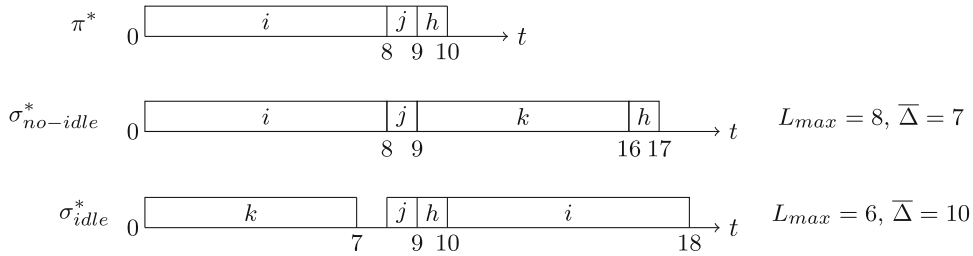
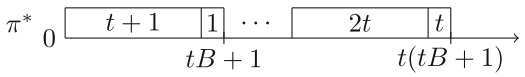


Fig. 6 Idle time insertion for the $1|\overline{\Delta} \leq \epsilon|L_{max}$ problem

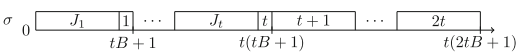
- $J^O = \{1, \ldots, 2t\}$
- $p_j = 1,\ j = 1, \ldots, t$
- $d_j = j(tB + 1),\ j = 1, \ldots, t$
- $p_j = tB,\ j = t + 1, \ldots, 2t$
- $d_j = t(2tB + 1),\ j = t + 1, \ldots, 2t$
- $J^N = \{2t + 1, \ldots, 5t\}$
- $p_j = ta_{j-2t},\ j = 2t + 1, \ldots, 5t$
- $d_j = t^2B + t - 1,\ j = 2t + 1, \ldots, 5t$
- $\epsilon = t^3B + t(t + 1)/2$

It is shown below, that a solution to the instance of $1|\overline{\Delta} \leq \epsilon,\ \overline{U} \leq 0|-$ exists if and only if a solution to the instance of 3-PARTITION exists.

Schedule π^* is constructed as follows.



If a solution to 3-PARTITION exists, there exist index sets $J_1, J_2, \ldots, J_t$, such that for any J_j, $\sum_{i \in J_j} a_i = B$ and $|J_j| = 3$. Then, the following schedule solves $1|\overline{\Delta} \leq \epsilon,\ \overline{U} \leq 0|-$.



On the other hand, in order to have a schedule with $\overline{U} \leq 0$, all jobs $t + 1, \ldots, 2t$ have to be scheduled after the new jobs, otherwise one of the new jobs would be late. Suppose that exactly h jobs of the set $\{1, \ldots, t\}$ are scheduled after the first job of $\{t+1, \ldots, 2t\}$ in σ^*. Each such job completes in σ^* no earlier than time $t^2B + tB + (t - h) + 1$, whereas it completes in π^* no later than time $t(tB+1)$. Moreover, jobs $t+1, \ldots, 2t$ are completed at times tB, $(2tB+1), \ldots, (t^2B+t-1)$ in π^* and are preceded by all new jobs and $t - h$ jobs of $\{1, \ldots, t\}$ and therefore are completed no earlier than times $(t^2B + (t - h) + tB), (t^2B + (t - h) + 2tB), \ldots, (t^2B + (t - h) + t^2B)$ in σ^*. Computing the total disruption, we obtain:

$\overline{\Delta} \geq t^3B + t(t - h) - t(t - 1)/2 + h(tB - h + 1)$
$= \epsilon + h(tB - h + 1)$.

Since $B \geq 3$, by definition of 3-PARTITION, we conclude that $h = 0$. Also, in order to have the disruption constraint satisfied, jobs $1, \ldots, t$ must be scheduled on time with respect to their initial completion times and the new jobs in sets $S_1, \ldots, S_t$ between them as shown in the figure above. The partition of the new jobs solves 3-PARTITION.

The same reduction also works for problem $1|\overline{\Delta} \leq \epsilon|\overline{T}$ and shows that 3-PARTITION reduces to the decision problem $1|\overline{\Delta} \leq \epsilon,\ \overline{T} \leq 0|-$.

In order to conclude the proof, we show that the problem is not a number problem. The first condition requires the existence of a polynomial λ of both the size of the rescheduling problem and the size of 3-PARTITION that bounds the largest number of the input, as follows:

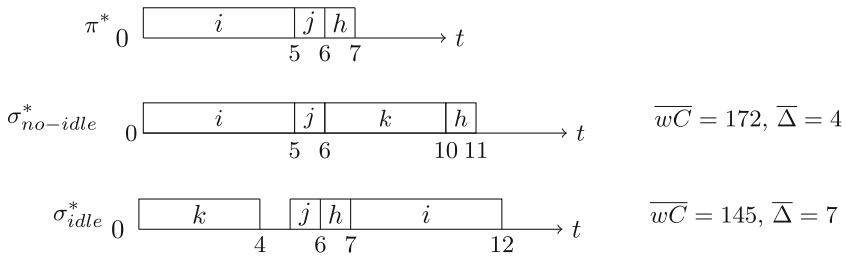


Fig. 7 Idle time insertion for the $1|\overline{\Delta} \leq \epsilon|\overline{wC}$ problem

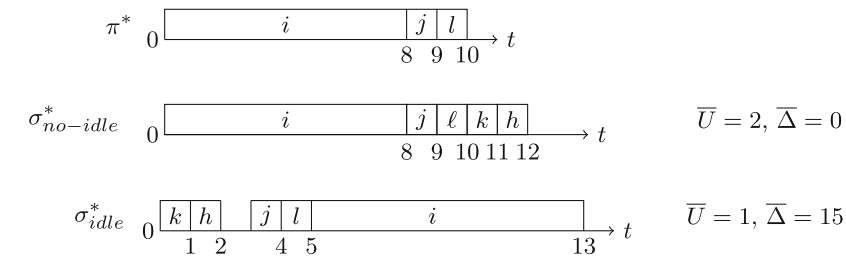


Fig. 8 Idle time insertion for the $1|\overline{\Delta} \leq \epsilon|\overline{U}$ and $1|\overline{\Delta} \leq \epsilon|\overline{T}$ problems

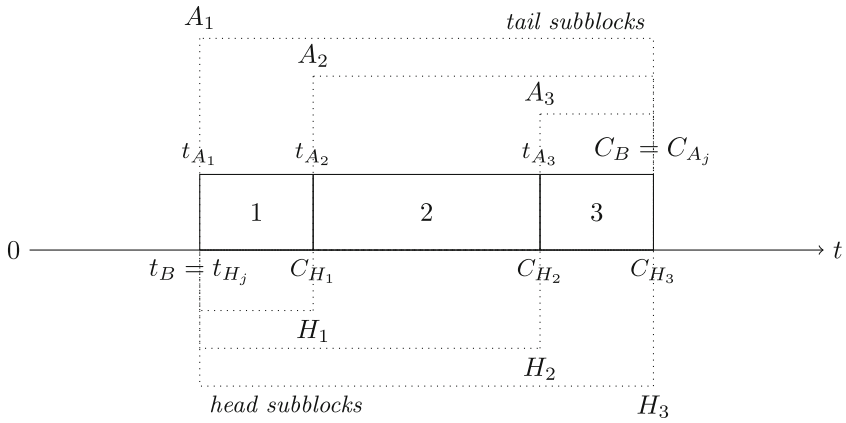


Fig. 9 A block B of jobs with three jobs, head subblocks H_j and tail subblocks A_j

$$\begin{aligned}&\max(\max_{j\in J^O\cup J^N}(p_j, d_j), \epsilon)\\ &\leq 2t^3B + t^2 + t \leq \lambda(2(|J^O| + |J^N|) + 1, t)\\ &\leq \lambda(10t + 1, t)\end{aligned}$$

Since 3-PARTITION is not a number problem, there exists a polynomial λ' such that $B \leq \lambda'(3t)$ which implies the existence of λ.

The second condition requires the existence of a polynomial ρ of the length of the size of the rescheduling instance that bounds the size of 3-PARTITION, as follows:

$$t \leq \rho(10t + 1)$$

Clearly, the statement holds and shows that the rescheduling problem is not a number problem.

3.2 Additional complexity results

Theorem 3 *Problem* $1|\overline{\Delta} \leq \epsilon|\overline{wC}$ *is* $\mathcal{NP}$*-hard.*

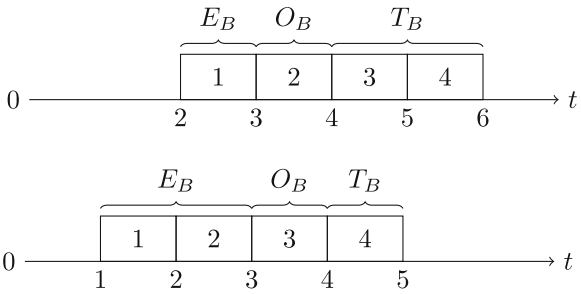


Fig. 10 A block B of jobs is shifted according to Algorithm 1

Proof Consider the special case with one old job j. In this case, we have $\overline{\Delta} = \Delta_j = \Delta_{max}$. Since problem $1|\Delta_{max} \leq \epsilon|\overline{wC}$ has been proven to be $\mathcal{NP}$-hard in the weak sense by Lendl et al. (2024), even when there is a single old job, problem $1|\overline{\Delta} \leq \epsilon|\overline{wC}$ is $\mathcal{NP}$-hard.

Theorem 4 *Problem* $1|\Delta_{max} \leq \epsilon|\overline{T}$ *is* $\mathcal{NP}$*-hard.*

Proof Consider the well-known generalization of the problem, i.e., the scheduling problem $1||\overline{T}$, which is known to be weakly $\mathcal{NP}$-hard. It is enough to take $\epsilon = +\infty$ to show that

$1||\overline{T}$ is a particular case of $1|\Delta_{max} \leq \epsilon|\overline{T}$, i.e., the rescheduling problem is $\mathcal{NP}$-hard.

4 Polynomial algorithms for fixed sequence rescheduling problems

From Tables 1 and 2, we can deduce that rescheduling problems that belong to the family of problems that need idle time insertion are all $\mathcal{NP}$-hard. In this section, we prove that the special case with fixed jobs sequence is polynomially solvable. The problem with fixed sequence becomes a timing problem, i.e., the problem of determining the exact starting times of the jobs. So, as soon as a sequence has to be evaluated, the timing problem must be solved. Besides the interest in the field of timing problems, these algorithms may be used for solution procedures such as in branch-and-bound algorithms, where each node is associated with a specific (possibly partial) sequence of jobs.

Assume we are given a fixed sequence of old and new jobs $\alpha = (1, 2, \ldots, n)$ that are initially scheduled with no inserted idle times. Notice, that we refer in this section to a sequence of jobs, rather than to a schedule, whenever we refer to the job permutation. On the contrary, a schedule is associated with both a job permutation, i.e., a sequence, and a list of job completion times. We also assume continuous completion times since it is a standard assumption when considering timing problems (see Chrétienne and Sourd (2003), Vidal et al. (2015)). In the following, we consider that jobs are indexed following their order in the sequence.

In the remaining part of the section, we first address timing problems with the maximum disruption constraint Δ_{max} before turning to the case of the total disruption $\overline{\Delta}$. Since timing problems usually determine optimal starting times of jobs in terms of time deviation from due dates, from now on we will call the original completion time of old jobs as d_j^C. So, we have $d_j^C = C_j(\pi^*)$.

4.1 Timing for problems $1|seq, \Delta_{max} \leq \epsilon|\{\overline{U}, \overline{T}\}$

This set of problems is identical to the set of problems of minimizing a regular objective function, subject to time window constraints. Given the value of the maximum disruption ϵ, for each old job we can define a release date $r_j = \min(0, d_j^C - \epsilon - p_j)$ and a deadline $\bar{d}_j = d_j^C + \epsilon$. While for any new job we set $r_j = 0$ and $\bar{d}_j = +\infty$. Clearly, processing a job outside the interval $[r_j, \bar{d}_j]$ makes the schedule infeasible with respect to the disruption constraint. The problem at hand is solved by the *minimum idle time policy* (Vidal et al., 2015) as follows. Each job of the sequence is scheduled at its earliest feasible execution time, i.e., right after the previous job completes or when it is released. If all jobs can be successfully scheduled in this way, from the first to the last, the schedule is feasible, and the increase in the objective function is minimal. The algorithm runs in $O(n)$ time.

4.2 Timing for problems $1|seq, \overline{\Delta} \leq \epsilon|\{L_{max}, \overline{wC}, \overline{U}, \overline{T}\}$

The timing algorithms we propose involve two steps: a pre-processing phase (see Sect. 4.2.1) and an iterated routine. The iterated routine is developed independently for the bottleneck objective function (see Sect. 4.2.2) and sum objective functions (see Sect. 4.2.3).

4.2.1 Pre-processing phase

The algorithm takes as input a sequence of jobs scheduled without idle times. If this schedule satisfies the constraint on the disruption, then the timing of jobs is optimal since any regular objective function is minimized in a compact schedule. Thus, we consider only the case where $\overline{\Delta} > \epsilon$.

The pre-processing answers the question of whether it is possible to get a feasible schedule without any increase in the current objective function. If such a schedule exists, it is also optimal since the current objective function value is minimum for the given job sequence. To answer the question, we minimize the total disruption subject to the constraint not to increase the current objective function: if the minimum disruption satisfies the threshold ϵ, we have a positive answer, otherwise, we have a negative answer, and the algorithm continues, with the iterated routine.

Call ω the initial value of the objective function in the compact schedule α. To solve the current problem, the new schedule σ needs to satisfy the constraint $f(\sigma) \leq \omega$. For this purpose, we assign deadlines $\tilde{d}_j$ to jobs, such that a violation of any of them implies an increase of f.

First, we focus on the case of function L_{max}. Given its value ω in the compact schedule, imposing $f_{max} \leq \omega$ is equivalent to imposing $f_j \leq \omega, \forall j \in J^O \cup J^N$. To have this constraint guaranteed, we set $\tilde{d}_j$ such that $\tilde{d}_j \leq d_j + \omega$. There are cases, where some job i has a more restrictive deadline than some job j that precedes i in the schedule. In this case, i will first meet its deadline if both jobs are right-shifted by the same amount, and j will never meet its own deadline if i is scheduled at a feasible time. Consequently, we set:

$$\tilde{d}_j = \begin{cases} d_j + \omega & j = n \\ \min(\tilde{d}_{j+1} - p_{j+1}, d_j + \omega) & j = 1, \ldots, n-1 \end{cases}$$

Now, let us consider the case of $\overline{f}$. The initial compact schedule α has an objective of ω, which is given by $\omega = \sum_{j \in J^O \cup J^N} f_j(\alpha)$. Set $\omega_j = f_j(\alpha), \forall j \in J^O \cup J^N$ as the

value of f_j in α. Since the sequence is fixed and the f_j's are regular functions, in any schedule σ with jobs in the same sequence as in α it holds $f_j(\sigma) \geq \omega_j$. But then, having $\overline{f} = \sum_{j \in J^O \cup J^N} f_j \leq \omega$ is equivalent to imposing $f_j \leq \omega_j$, $\forall j \in J^O \cup J^N$.

We illustrate the above definitions on the objective functions we are considering in this paper. When $\overline{f}$ is considered to be $\overline{U}$,

$$\tilde{d}_j = \begin{cases} d_j & \omega_j = 0,\ j = n \\ +\infty & \omega_j = 1,\ j = n \\ \min(\tilde{d}_{j+1} - p_{j+1}, d_j) & \omega_j = 0,\ j = 1, \ldots, n-1 \\ \min(\tilde{d}_{j+1} - p_{j+1}, +\infty) & \omega_j = 1,\ j = 1, \ldots, n-1 \end{cases}$$

When $\overline{f}$ is considered to be $\overline{T}$,

$$\tilde{d}_j = \begin{cases} d_j & \omega_j = 0,\ j = n \\ d_j + \omega_j & \omega_j = 1,\ j = n \\ \min(\tilde{d}_{j+1} - p_{j+1}, d_j) & \omega_j = 0,\ j = 1, \ldots, n-1 \\ \min(\tilde{d}_{j+1} - p_{j+1}, d_j + \omega_j) & \omega_j = 1,\ j = 1, \ldots, n-1 \end{cases}$$

Finally, notice that when we are considering $\overline{f}$ to be $\overline{wC}$, $\tilde{d}_j = \omega_j$, i.e., there is no way to shift jobs without increasing the objective function and the pre-processing will not modify the initial compact schedule.

The pre-processing algorithm minimizes $\overline{\Delta}$, given the limit on f and checks if $\overline{\Delta} \leq \epsilon$. The idea behind the algorithm follows the one for solving the minimum earliness/tardiness scheduling problem with a fixed sequence by Garey et al. (1988) that runs in $O(n \log n)$ time. Minimizing the sum of the earliness and tardiness over a job set J w.r.t due dates d_j means, by definition, minimizing $\sum_{j \in J} |C_j - d_j|$, which is equivalent to minimizing the total disruption w.r.t. the original completion times. So, we modify the algorithm of Garey et al. (1988) to minimize the total disruption subject to job deadline constraints. We introduce some useful notation and illustrate the algorithm in the following paragraphs.

At any point of the schedule, let a block be a group of jobs that are scheduled consecutively without interruptions and preceded and followed by idle times, with the exception of the block starting at time 0: in this case, it is only followed by an idle time. A subblock is any job sequence of consecutive jobs that is part of a block. A subblock is a *head* if it is preceded by an idle time or it starts at time 0. A subblock is a *tail* if it is followed by idle times or ends the schedule. A schedule σ is a collection of blocks $B \in \mathcal{B}$ separated by idle times. Each block $B \in \mathcal{B}$ has a starting time t_B and a completion time C_B. Figure 9 shows an example of a block made of three jobs $j = 1, 2, 3$, constituted by head subblocks H_j with starting times $t_{H_j} = t_B$ and completion times C_{H_j} and tail subblocks A_j with starting times t_{A_j} and completion times C_{A_j}, for every job $j = 1, 2, 3$.

We say that a job $j \in J^O$ in the rescheduling problem is *early*, *on time* or *late* if it completes before, exactly at or after d_j^C. Notice that the new jobs have no due date d_j^C as the total disruption is only computed on J^O. For a given block or subblock B in a schedule σ, let us denote by $E_B(\sigma)$ the set of early old jobs, by $O_B(\sigma)$ the set of on time old jobs and by $T_B(\sigma)$ the set of tardy old jobs (E_B, O_B, T_B when there is no ambiguity). Notice that if a tardy or an on time job is delayed by one time unit, then the disruption of the considered job increases by one unit too. Similarly, the disruption of an early or on time job increases by one unit for each time unit the job is brought forward. For instance, delaying a block with $|E_B| - |O_B| - |T_B| < 0$ increases the total disruption, and bringing forward a block with $|E_B| + |O_B| - |T_B| > 0$ does it too.

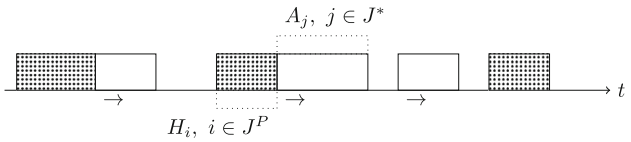


Fig. 11 Set of tail subblocks $A_j,\ j \in J^*$ (resp. head subblocks $H_i,\ i \in J^P$) that shift (resp. stay)

The algorithm for the earliness/tardiness problem, iteratively adds a job following the order of the fixed sequence, appending it to the last block or creating a new block. This job is either inserted at its due date, if this does not cause an overlapping of jobs, or exactly next to the previously added job. Let B be the block to which job j belongs after the insertion: if $|E_B| + |O_B| - |T_B| = 0$, then the block is shifted to the left until a job of the block becomes on time or the previous block is met or $t_B = 0$.

Algorithm 1 Algorithm for $1|seq, f \leq \omega|\overline{\Delta}$

```
Compute jobs deadlines following equations (1), (2), (3), (4)
C_1 = min(max(p_1, d_1^C), d̃_1)
B := 1
for i = 1, ..., n − 1 do
    if i + 1 ∈ J^O then
        if C_i + p_{i+1} ≤ min(d_{i+1}^C, d̃_{i+1}) then
            C_{i+1} = min(d_{i+1}^C, d̃_{i+1})
            B := B + 1
        else
            C_{i+1} = C_i + p_{i+1}
            if |E_B| + |O_B| − |T_B| = 0 then
                left-shift B by min(min_{j∈J^O∩B, d_j^C<C_j}(C_j − d_j^C), t_B − C_{B−1}, t_B)
            end if
        end if
    else
        C_{i+1} = C_i + p_{i+1}
    end if
end for
```

To adapt this algorithm to our problem, we distinguish the steps done for the set of new jobs and those for the set of old

jobs. Considering new jobs, since they do not have any impact on the total disruption, we can schedule them immediately after their predecessors or at $t = 0$ if there is no job scheduled before, i.e., for any job $i \in J^N$ we define its completion time as $C_i = C_{i-1} + p_i$ with $C_0 = 0$. If we consider old jobs, we schedule them so that they are completed by d_j^C, or at $\tilde{d}_j$ if $\tilde{d}_j < d_j^C$. If scheduling a job by d_j^C causes overlapping of jobs, then the current job is scheduled exactly next to the previously added job. Notice that for any job that is appended to its predecessor, the deadlines are satisfied by definition. The modified algorithm is shown in Algorithm 1.

We sketch the situation of a block moved during the algorithm (lines 11–12 of Algorithm 1) with an example represented in Fig. 10. In the example, there are 4 jobs $j = 1, \ldots, 4$ with unit processing times. The jobs have, respectively, $d_1^C = 5$, $d_2^C = 4$, $d_3^C = 1$, $d_4^C = 2$ and have, respectively, $\tilde{d}_1 = 3$, $\tilde{d}_2 = 4$, $\tilde{d}_3 = \tilde{d}_4 = 6$. In this way, jobs 1, 2, 3 are scheduled consecutively until job 4 is appended. At this point, all jobs are left-shifted by one time unit, so that job 3 goes on time, and the balance between early, on time, and tardy jobs is restored.

The case depicted in Fig. 10 helps to introduce the following lemma.

Lemma 1 *In any schedule returned by Algorithm 1, the following two conditions hold:*

1. *In any head subblock H_j that is not starting at time 0 (see Fig. 9), we have $|E_{H_j}| + |O_{H_j}| - |T_{H_j}| > 0$.*
2. *In any tail subblock A_j (see Fig. 9), where none of the jobs has $C_j = \tilde{d}_j$, we have $|E_{A_j}| - |O_{A_j}| - |T_{A_j}| \leq 0$.*

Proof First, consider condition 1. If H_j is the first block scheduled, and it holds, at some iteration, $|E_{H_j}| + |O_{H_j}| - |T_{H_j}| = 0$, then it is left-shifted until $|E_{H_j}| + |O_{H_j}| - |T_{H_j}| > 0$, or it meets time 0. If H_j is the second block, and it holds, at some iteration, $|E_{H_j}| + |O_{H_j}| - |T_{H_j}| = 0$, then it is left-shifted until whether $|E_{H_j}| + |O_{H_j}| - |T_{H_j}| > 0$, or it is merged with a block that starts at 0. By repeating this reasoning, if H_j is any other following block, and it holds, at some iteration, $|E_{H_j}| + |O_{H_j}| - |T_{H_j}| = 0$, then it is left-shifted as a whole until one of the following conditions are met: it holds $|E_{H_j}| + |O_{H_j}| - |T_{H_j}| > 0$; it is merged with a block that starts at 0; or it is merged with another block H_i with $|E_{H_i}| + |O_{H_i}| - |T_{H_i}| > 0$. For the latter case, for the new merged block H'_j, we have $|E_{H'_j}| + |O_{H'_j}| - |T_{H'_j}| = |E_{H_i}| + |O_{H_i}| - |T_{H_i}| + |E_{H_j}| + |O_{H_j}| - |T_{H_j}| > 0$. So, we can conclude that any head subblock H_j in the final schedule has $|E_{H_j}| + |O_{H_j}| - |T_{H_j}| > 0$ or it starts at time 0.

Lets consider now condition 2. Consider a block A_j, starting with job j. If at any point of the algorithm, the block A_j is left-shifted, then it must have $|E_{A_j}| + |O_{A_j}| - |T_{A_j}| = 0$ (condition 1). After the shift, the new set E'_{A_j} of early jobs is then made up of $E_{A_j} \cup O_{A_j}$ and the set of tardy jobs contains the new sets of tardy, resp. on time jobs, i.e., T'_{A_j} resp. O'_{A_j}, so that $|E'_{A_j}| - |O'_{A_j}| - |T'_{A_j}| = 0$. Notice that the shift in the current iteration of the algorithm stops by definition before any of the tardy jobs becomes early. Consider now the job that is scheduled at the next iteration, and call this job i. Job i is appended to block A_j if one of the two following conditions hold: $d_i^C \leq C_{A_j} + p_i$, and in this case job i is tardy or on time, or $\tilde{d}_i = C_{A_j} + p_i$. It follows that in the new block A'_j (block A_j with the newly appended job i), we have $|E_{A'_j}| - |O_{A'_j}| - |T_{A'_j}| > 0$ only if job i is early, and $C_i = \tilde{d}_i$. Otherwise, $|E_{A'_j}| - |O_{A'_j}| - |T_{A'_j}| \leq 0$. We can repeat this argument for all generated blocks. Notice that if two blocks merge at some iteration, then, whether the inequality holds for both blocks, or there is at least an early job scheduled at its deadline in the block scheduled first.

Theorem 5 *Algorithm 1 returns an optimal solution for the problem* $1|seq, f \leq \omega|\overline{\Delta}$.

Proof Assume by contradiction that there exists a schedule σ', with $f \leq \omega$ and smaller $\overline{\Delta}$ than that of schedule σ^* produced by Algorithm 1. Notice that, in the following, we use the fact that the disruption increases by the same amount of shift by which a tardy or on time job is delayed. Similarly, the disruption increases by the same amount of shift by which an early or an on time job is brought forward.

If there exists such a schedule σ', there is at least a subblock B, with at least one old job, that is scheduled at a different starting time than in σ^*.

On one hand, it can be $t_B(\sigma') < t_B(\sigma^*)$. Since jobs cannot overlap, B must be a head subblock (see subblocks H_j in Fig. 9). Given Lemma 1, in σ^*, it holds that $|E_{B(\sigma^*)}| + |O_{B(\sigma^*)}| - |T_{B(\sigma^*)}| > 0$. Then, shifting the subblock to the left increases the total disruption by $|E_{B(\sigma^*)}| + |O_{B(\sigma^*)}| - |T_{B(\sigma^*)}| > 0$ for each unit of time, i.e., $\overline{\Delta}(\sigma') > \overline{\Delta}(\sigma^*)$, and $f(\sigma') = f(\sigma^*)$.

On the other side, it can be $t_B(\sigma') > t_B(\sigma^*)$. Since jobs cannot overlap, B must be a tail subblock (see subblocks A_j in Fig. 9). Also, if $t_B(\sigma') > t_B(\sigma^*)$ is feasible, for all jobs, we have $C_j < \tilde{d}_j$. Then, from Lemma 1, we have $|E_{B(\sigma^*)}| - |O_{B(\sigma^*)}| - |T_{B(\sigma^*)}| \leq 0$. Hence, shifting the subblock to the right increases the total disruption by $|O_{B(\sigma^*)}| + |T_{B(\sigma^*)}| - |E_{B(\sigma^*)}| > 0$ for each unit of time, i.e., $\overline{\Delta}(\sigma') \geq \overline{\Delta}(\sigma^*)$ and $f(\sigma') = f(\sigma^*)$.

Consequently, σ' never dominates σ^*.

Consider schedule σ returned by Algorithm 1. Let us introduce, for each tail subblock A_j, $j = 1, \ldots, n$, a value

$$\mu_j = \begin{cases} |E_{A_j}| - |O_{A_j}| - |T_{A_j}| & \forall j \in J^O \\ 0 & \forall j \in J^N \end{cases}$$

The value μ_j is the change in the total disruption if the tail subblock A_j is right-shifted by one time unit. Since only the old jobs contribute to a change of the disruption, we assign a value $\mu_j = 0$ to each A_j starting with a new job (these tail subblocks are not relevant for modifying the total disruption because it is enough to consider the tail subblock starting with the first old job scheduled after j to obtain the same disruption change). If $\mu_j > 0$, the total disruption decreases when A_j is right-shifted, and vice versa.

Recall that the jobs are indexed by position in the sequence. For any block B, we call j^* the job in B with $\mu_{j^*} > 0$, such that, for any other job $i < j^*$ that belongs to B, $\mu_{j^*} \geq \mu_i$. Notice, that by definition, j^* is always an old job or does not exist. Call J^* the index set that contains all j^* of a given schedule. For each block B that does not start with a job $j \in J^*$, call s_B the job that precedes job $j \in J^*$ in the same block, if such j exists, or the job that ends the block, otherwise. Call J^P the index set that contains all s_B of a given schedule.

Lemma 2 *Right shifting the head subblocks $H_i,\ i \in J^P$ never leads to a decrease of the total disruption.*

Proof Consider any block that contains a job $i \in J^P$ and a job $j \in J^*$. Call k the first job of the block. Since $\mu_k \leq \mu_j$ by definition of j, then for the head subblock H_i we have $|E_{H_i}| - |O_{H_i}| - |T_{H_i}| = \mu_k - \mu_j \leq 0$. Then, right shifting H_i alone does not decrease the total disruption.

Lemma 3 *Right shifting any of the tail subblocks with $\mu_j > 0$ always increases f.*

Proof By Lemma 1, for any tail subblock A_j, if $|E_{A_j}| - |O_{A_j}| - |T_{A_j}| = \mu_j > 0$, at least one job in A_j has $C_j = \tilde{d}_j$. Then, right shifting the subblock always increases f.

In the next two sections, we establish how to iteratively identify the optimal set of subblocks to be shifted to determine the optimal timing, first for problem $1|seq, \overline{\Delta} \leq \epsilon|L_{max}$, next for problem $1|seq, \overline{\Delta} \leq \epsilon|\{\overline{wC}, \overline{U}, \overline{T}\}$.

4.2.2 Timing for problem $1|seq, \overline{\Delta} \leq \epsilon|L_{max}$

We proved that right shifting any of the tail subblocks with corresponding $\mu_j > 0$, always leads to an increase of any objective function f (Lemma 3) meaning that L_{max} precisely increases by the amount of the shift (the maximum lateness increases proportionally with the increase of the job completion times). Considering this, it is more convenient to right shift simultaneously the subblocks $A_j,\ j \in J^*$, to obtain the maximum unitary gain in the total disruption for each unitary increase of L_{max}. The situation is depicted in Fig. 11. The white blocks are tail subblocks $A_j,\ j \in J^*$, while the dotted blocks are the head subblocks, which can be denoted with $H_i,\ i \in J^P$ (recall the notation introduced in the previous section), and that do not move.

The total unit gain μ in the total disruption per unitary time shift is given by $\sum_{j \in J^*} \mu_j$. We can derive the following corollary of Lemmas 2 and 3 for the $1|seq, \overline{\Delta} \leq \epsilon|L_{max}$ problem.

Corollary 1 *In any schedule σ, it is necessary and sufficient to shift all the tail subblocks $A_j,\ j \in J^*$ of schedule σ to find a schedule with the minimum increase of L_{max} and the largest decrease of $\overline{\Delta}$ for each unit of inserted idle time.*

Algorithm 2 Algorithm for problem $1|seq, \overline{\Delta} \leq \epsilon|L_{max}$ (ALG_M)

```
Run Algorithm 1 on the given job sequence and store solution in σ
τ = Δ̄(σ) − ε
if τ ≤ 0 then
    return σ
end if
Compute J*, μ, ρ, λ
while μ > 0 do
    δ = min(τ/μ, ρ, λ)
    Shift all tail subblocks A_j, j ∈ J* by δ
    τ := τ − δ · μ
    if τ ≤ 0 then
        return σ
    end if
    Compute J*, μ, ρ, λ
end while
return The given job sequence is infeasible
```

The algorithm iteratively shifts all the tail subblocks $A_j,\ j \in J^*$. When moving these subblocks several events may happen:

(E_1). One block meets another block.

(E_2). At least one old job reaches its original completion time d_j^C.

Consider event E_1. The event occurs when at least a tail subblock $A_j,\ j \in J^*$ is followed by a head block $H_i,\ i \in J^P$. Then, E_1 occurs after

$$\rho = \min_{\substack{A_j,\ H_i: \\ i>j,\ j \in J^*,\ i \in J^P}} (t_{H_i} - C_{A_j})$$

units of time.

Next, consider event E_2. The first event of this type occurs after

$$\lambda = \min_{\substack{j \in A_i: \\ i \in J^*,\ d_j^C > C_j}} (d_j^C - C_j)$$

units of time.

Whenever one of the mentioned events occurs, to restore the optimality condition of Corollary 1, the μ_j values must

be recomputed to find the new optimal set of tail subblocks $A_j,\ j \in J^*$ to be shifted.

There are two stopping conditions for the algorithm. First, the algorithm stops as soon as feasibility is reached, i.e., if at any iteration we can shift the tail subblocks $A_j,\ j \in J^*$, by $\frac{\tau}{\mu}$ units of time before any of the events occur. Secondly, the algorithm stops when there are no strictly early tail subblocks. In this case, all $\mu_j \leq 0$ and there is no way to reach feasibility with the given sequence of jobs.

Putting everything together, we obtain *ALG_M* (Algorithm 2). The algorithm takes as input the schedule computed by Algorithm 1. Then, at each iteration the algorithm computes the index set J^*, the corresponding $\mu = \sum_{j \in J^*} \mu_j$ and when the next event occurs (quantities $\lambda,\ \rho$). Next, all the subblocks $A_j,\ j \in J^*$ are right-shifted (see Fig. 11). If at any time, feasibility is reached or index set J^* is empty, i.e., $\mu \leq 0$, then the algorithm stops.

Lemma 4 *Algorithm ALG_M finds an optimal solution for* $1|seq, \overline{\Delta} \leq \epsilon|f_{max}$, *with* f_{max} *given by* L_{max}, *in* $O(n^2)$ *time.*

Proof Computing $J^*,\ \mu,\ \rho$ and $\lambda,\ \forall j \in J^O \cup J^N$ can be done in $O(n)$ time by computing the μ_j's backward and meanwhile keeping track of $\mu,\ \rho$ and λ and $j \in J^*$. Starting from the last job of each block, we assign $\mu_j = 1$ if $d_j^C - C_j > 0, \mu_j = -1$ if $d_j^C - C_j \leq 0$ or $\mu_j = 0$ if j is a new job. Then, for each further job in the same block, $\mu_j = \mu_{j+1} + 1$ if $d_j^C - C_j > 0, \mu_j = \mu_{j+1} - 1$ if $d_j^C - C_j \leq 0$ or $\mu_j = \mu_{j+1}$ if j is a new job.

Next, we bound the number of events, i.e., the number of iterations of the algorithm by an additional factor $O(n)$. The number of E_2 events is bounded by n, since once an old job goes on time, it cannot become early again. To bound the number of E_1 events, consider what happens after one occurs. Two blocks $A_j,\ j \in J^*$ and $A_i,\ i \notin J^*$, with $i > j$, merge and possibly continue to shift. However, the two blocks will not split again, since, by definition any block is also a tail block and $|E_{A_i}| - |O_{A_i}| - |T_{A_i}| \leq 0$ (Lemma 1), so right shifting A_i alone never decreases the total disruption. Then, the number of merging operations, i.e., the number of E_1 events, is bounded by the number of blocks, which is at most n.

The time complexity of Algorithm 2 is then bounded by $O(n^2)$ time.

4.2.3 Timing for problems $1|seq, \overline{\Delta} \leq \epsilon|\{\overline{wC}, \overline{U}, \overline{T}\}$

We now set the procedure to identify and shift subblocks to find optimal timing for objectives $\overline{f} \in \{\overline{wC}, \overline{U}, \overline{T}\}$. For any tail subblock A_k, let us introduce a value δ_k^f as the increase of $\overline{f}$ when right shifting A_k by one time unit. Let A_j be the tail subblock with $\mu_j > 0$ and $\mu_j/\delta_j^f = \max_{i \in J^*}(\mu_i/\delta_i^f)$. We break ties by considering the block with the largest index. Let $\overline{\Delta}_{curr}$ be the total disruption of the current schedule. Let $\tau_j > 0$ be the maximum decrease of the total disruption that can be obtained if A_j is right-shifted such that $\tau_j \leq \overline{\Delta}_{curr} - \epsilon$. Call $\delta_j = \tau_j/\mu_j$ the corresponding amount of shift. Notice that the case where $\overline{\Delta}_{curr} \leq \epsilon$ is excluded because in this case we have a feasible schedule.

Lemma 5 *In any schedule* σ, *it is optimal to iteratively identify and right shift the tail subblock* A_j *by a quantity* δ_j *to find a schedule with the minimum increase of* $\overline{f} \in \{\overline{wC}, \overline{U}, \overline{T}\}$.

Proof In any schedule σ, the increase of the objective function obtained by shifting A_j is given by $\delta_j \cdot \delta_j^f$. Assume that there exists another subblock A_i that leads to a smaller increase of the objective function and that is obtained with a shift $\delta_i = \tau_i/\mu_i$. We can write:

$$\frac{\tau_i}{\mu_i}\delta_i^f \leq \frac{\tau_j}{\mu_j}\delta_j^f$$

By definition of A_j, we have $\delta_j^f/\mu_j \leq \delta_i^f/\mu_i$, therefore it must hold $\tau_i \leq \tau_j$. We can obtain the overall decrease τ_j of the total disruption by shifting A_j or first shifting A_i by δ_i and then A_j by $\tau_j - \tau_i$. We can compare the corresponding increase of the objective function in these two cases:

$$\frac{\tau_j}{\mu_j}\delta_j^f - \frac{\tau_i}{\mu_i}\delta_i^f - \frac{(\tau_j - \tau_i)}{\mu_j}\delta_j^f = \frac{\tau_i}{\mu_j}\delta_j^f + \frac{(\tau_j - \tau_i)}{\mu_j}\delta_j^f - \frac{\tau_i}{\mu_i}\delta_i^f - \frac{(\tau_j - \tau_i)}{\mu_j}\delta_j^f \leq 0$$

which means that is always better to right shift A_j rather than A_i.

The main difference with problem $1|seq, \overline{\Delta} \leq \epsilon|L_{max}$ is that here, each time, only one tail subblock is shifted, instead of moving multiple tail subblocks simultaneously. The algorithm for $1|seq, \overline{\Delta} \leq \epsilon|\overline{f}$, referred to as *ALG_S*, iteratively shifts the optimal tail subblock A_j, that we call A_j^* in the following (with the corresponding increase of the total disruption for a unitary right shift μ_j^*), until at least an event occurs. In addition to events (E_1) and (E_2) from the previous Sect. 4.2.2, we introduce the following event:

(E_3). At least one job reaches its due date d_j.

The additional event E_3 may occur, when function $\overline{f}$ depends on due dates, i.e., for $\overline{U}$ and $\overline{T}$, and causes a change in the contribution δ_j^f to the objective function for a unitary shift, when moving a subblock. The first event of this type occurs after $\iota = \min_{j \in J^O \cup J^N, d_j > C_j}(d_j - C_j)$ units of time.

ALG_S is sketched in Algorithm 3.

Lemma 6 *Algorithm ALG_S finds an optimal solution for any problem* $1|seq, \overline{\Delta} \leq \epsilon|\overline{f}$, *with* $\overline{f} \in \{\overline{wC}, \overline{U}, \overline{T}\}$, *in* $O(n^2)$ *time.*

Algorithm 3 Algorithm for problem $1|seq, \overline{\Delta} \leq \epsilon|\overline{f}$ (*ALG_S*)

1: Run Algorithm 1 on the given job sequence and store solution in σ
2: $\tau = \overline{\Delta(\sigma)} - \epsilon$
3: **if** $\tau \leq 0$ **then**
4: **return** σ
5: **end if**
6: Compute $A_j^*,\ \mu_j^*,\ \rho,\ \lambda, \iota$
7: **while** $\mu_j^* > 0$ **do**
8: $\delta = \min(\frac{\tau}{\mu_j^*}, \rho, \lambda, \iota)$
9: shift A_j^* by δ
10: $\tau := \tau - \delta \cdot \mu_j^*$
11: **if** $\tau \leq 0$ **then**
12: **return** σ
13: **end if**
14: Compute $A_j^*,\ \mu_j^*,\ \rho,\ \lambda, \iota$
15: **end while**
16: **return** The given job sequence is infeasible

Proof At each iteration, the algorithm identifies A_j^* and computes $\mu_j^*,\ \rho,\ \lambda, \iota$. Everything can be computed in $O(n)$ time.

Next, we bound the number of events. We have already shown that the number of E_2 events is bounded by n (proof of Lemma 4). In the same way, the number of E_3 events is bounded by n, since this type of event is generated by a job meeting its due date and there are n jobs. To bound the number of E_1 events, consider what happens after one occurs. We consider an E_1 event, where no event E_2 or E_3 occurs. If two blocks A_j, resp. A_i, merge then it must hold that before the merging operation, we have $A_j = A_j^*$ and, by definition, $\frac{\mu_j}{\delta_j^f} \geq \frac{\mu_i}{\delta_i^f}$. After the merge, if there are no events E_2 or E_3 happening at the same time, then the value $\frac{\mu_j}{\delta_j^f}$ in subblock A_j does not change if we do not consider the contribution of the merged block A_i. Then, the $\frac{\mu_\ell}{\delta_\ell^f}$ ratio for the merged block $A_j \cup A_i$, that will be used at the next iteration is given by $\frac{\mu_j+\mu_i}{\delta_j^f+\delta_i^f}$. If any of the two subblocks, among the merged block and subblock A_i, has to be shifted next, then it must be the merged block because we have

$$\frac{\mu_j + \mu_i}{\delta_j^f + \delta_i^f} - \frac{\mu_i}{\delta_i^f} = \frac{\delta_i^f \mu_j + \delta_i^f \mu_i - \delta_j^f \mu_i - \delta_i^f \mu_i}{\delta_i^f (\delta_j^f + \delta_i^f))} = \frac{\delta_i^f \mu_j - \delta_j^f \mu_i}{\delta_i^f (\delta_j^f + \delta_i^f))} \geq 0. \quad (1)$$

In Eq. (1), we use the fact that $\frac{\mu_j}{\delta_j^f} \geq \frac{\mu_i}{\delta_i^f}$, and that the denominator is always positive since we consider regular objective functions.

This proves that, once two blocks merge, they do not split again unless another type of event is involved. The total number of E_1, E_2, and E_3 events is then bounded by $3n$.

Given the maximum total number of iterations, i.e., the total number of events, and the number of operations for updating the values to identify the optimal idle time insertion, the overall time complexity of the algorithm is in $O(n^2)$ time.

5 Conclusions

We considered the set of rescheduling problems for new orders on a single machine, that arise from the combination of several scheduling objective functions and two disruption criteria. The new jobs have to be integrated into the original optimal schedule of old jobs to minimize the scheduling objective function, subject to a constraint on disruption. We considered the most commonly used scheduling objective functions, i.e., total completion time, total weighted completion time, total number of late jobs, total tardiness and maximum lateness, and two disruption criteria, i.e., the total and maximum absolute time deviation of old jobs.

The first contribution is a complete classification of problems according to their complexity and need for machine idle time insertion. In particular, the two disruption criteria are shown to have different properties, and the relevance of whether or not to preserve the original order of the old jobs with respect to the insertion of idle time is highlighted.

The second set of results is presented in the form of three polynomial-time timing algorithms for solving the $\mathcal{NP}$-hard rescheduling problems that may require the insertion of idle times, when the order of jobs is given.

This work aims to provide a reference framework for solving a whole range of rescheduling problems. Therefore, future research should first focus on the development and testing of efficient exact algorithms for problems that may require the insertion of idle times. We suggest the use of timing algorithms in solution procedures that exploit branch-and-bound techniques. In addition, the research should be extended to different machine environments and should be addressed both with exact algorithms and with heuristics for practical applications.

Funding Open access funding provided by University of Bern

Declarations

Conflict of interest The authors have no conflict of interest to declare that are relevant to the content of this article.

References

Aytug, H., Lawley, M. A., McKay, K., et al. (2005). Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research, 161*(1), 86–110. https://doi.org/10.1016/j.ejor.2003.08.027

Chrétienne, P., & Sourd, F. (2003). PERT scheduling with convex cost functions. *Theoretical Computer Science, 292*(1), 145–164. https://doi.org/10.1016/S0304-3975(01)00220-1

Fang, K., Luo, W., Pinedo, M. L., et al. (2023). Rescheduling for new orders on a single machine with rejection. *Journal of the Operational Research Society*. https://doi.org/10.1080/01605682.2023.2197000

Garey, M. R., Tarjan, R. E., & Wilfong, G. T. (1988). One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research, 13*(2), 330–348. https://doi.org/10.1287/moor.13.2.330

Graham, R., Lawler, E., Lenstra, J., et al. (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

Guo, Y., Huang, M., Wang, Q., et al. (2021). Single-machine rework rescheduling to minimize total waiting time with fixed sequence of jobs and release times. *IEEE Access, 9*, 1205–1218. https://doi.org/10.1109/ACCESS.2019.2957132

Hall, N., & Potts, C. (2004). Rescheduling for new orders. *Operations Research, 52*(3), 440–453. https://doi.org/10.1287/opre.1030.0101

Hall, N., Liu, Z., & Potts, C. (2007). Rescheduling for multiple new orders. *INFORMS Journal on Computing, 19*(4), 633–645. https://doi.org/10.1287/ijoc.1060.0209

Lendl, S., Pferschy, U., & Rener, E. (2024). Rescheduling with new orders under bounded disruption. *INFORMS Journal on Computing*. https://doi.org/10.1287/ijoc.2023.0038

Liu, L., & Zhou, H. (2015). single-machine rescheduling with deterioration and learning effects against the maximum sequence disruption. *International Journal of Systems Science, 46*(14), 2640–2658. https://doi.org/10.1080/00207721.2013.876519

Liu, Z., Lu, L., & Qi, X. (2018). Cost allocation in rescheduling with machine unavailable period. *European Journal of Operational Research, 266*(1), 16–28. https://doi.org/10.1016/j.ejor.2017.09.015

Luo, W., Chin, R., Cai, A., et al. (2022). A tardiness-augmented approximation scheme for rejection-allowed multiprocessor rescheduling. *Journal of Combinatorial Optimization, 44*, 690–722. https://doi.org/10.1007/s10878-022-00857-y

Pferschy, U., Resch, J., & Righini, G. (2023). Algorithms for rescheduling jobs with a LIFO buffer to minimize the weighted number of late jobs. *Journal of Scheduling, 26*, 267–287. https://doi.org/10.1007/s10951-022-00751-9

Rener, E., Salassa, F., & T'kindt, V. (2022). Single machine rescheduling for new orders with maximum lateness minimization. *Computers & Operations Research, 144*, 105815. https://doi.org/10.1016/j.cor.2022.105815

Teghem, J., & Tuyttens, D. (2014). A bi-objective approach to reschedule new jobs in a one machine model. *International Transactions in Operational Research, 21*(6), 871–898. https://doi.org/10.1111/itor.12066

Vidal, T., Crainic, T. G., Gendreau, M., et al. (2015). Timing problems and algorithms: Time decisions for sequences of activities. *Networks, 65*(2), 102–128. https://doi.org/10.1002/net.21587

Vieira, G., Herrmann, J., & Lin, E. (2003). Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *Journal of Scheduling, 6*(1), 39–62. https://doi.org/10.1023/A:1022235519958

Wang, D., Yin, Y., & Cheng, T. (2018). Parallel-machine rescheduling with job unavailability and rejection. *Omega, 81*, 246–260. https://doi.org/10.1016/j.omega.2018.04.008

Yang, B. (2007). Single machine rescheduling with new jobs arrivals and processing time compression. *International Journal of Advanced Manufacturing Technologies, 34*, 378–384. https://doi.org/10.1007/s00170-006-0590-7

Yuan, J., & Mu, Y. (2007). Rescheduling with release dates to minimize makespan under a limit on the maximum sequence disruption. *European Journal of Operational Research, 182*(2), 936–944. https://doi.org/10.1016/j.ejor.2006.07.026

Yuan, J., Mu, Y., Lu, L., et al. (2007). Rescheduling with release dates to minimize total sequence disruption under a limit on the makespan. *European Journal of Operational Research, 24*(6), 789–796. https://doi.org/10.1142/S021759590700153X

Zhang, X., Lin, W. C., & Wu, C. C. (2022). Rescheduling problems with allowing for the unexpected new jobs arrival. *Journal of Combinatorial Optimization, 43*, 630–645. https://doi.org/10.1007/s10878-021-00803-4

Zhao, C., & Tang, H. (2010). Rescheduling problems with deteriorating jobs under disruptions. *Applied Mathematical Modelling, 34*(1), 238–243. https://doi.org/10.1016/j.apm.2009.03.037

Zhao, Q., Lu, L., & Yuan, J. (2016). Rescheduling with new orders and general maximum allowable time disruptions. *4OR, 14*, 261–280. https://doi.org/10.1007/s10288-016-0308-0