




# A tardiness-augmented approximation scheme for rejection-allowed multiprocessor rescheduling

Wenchang Luo<sup>1</sup> · Rylan Chin<sup>2</sup> · Alexander Cai<sup>2</sup> · Guohui Lin<sup>2</sup>  · Bing Su<sup>3</sup> · An Zhang<sup>4</sup>

Accepted: 21 March 2022

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

## Abstract

In the multiprocessor scheduling problem to minimize the total job completion time, an optimal schedule can be obtained by the shortest processing time rule and the completion time of each job in the schedule can be used as a guarantee for scheduling revenue. However, in practice, some jobs will not arrive at the beginning of the schedule but are delayed and their delayed arrival times are given to the decision-maker for possible rescheduling. The decision-maker can choose to reject some jobs in order to minimize the total operational cost that includes three cost components: the total rejection cost of the rejected jobs, the total completion time of the accepted jobs, and the penalty on the maximum tardiness for the accepted jobs, for which their completion times in the planned schedule are their virtual due dates. This novel rescheduling problem generalizes several classic NP-hard scheduling problems. We first design a

---

✉ Guohui Lin  
guohui@ualberta.ca

✉ Bing Su  
subing684@sohu.com

Wenchang Luo  
luowenchang@163.com

Rylan Chin  
rchin@ualberta.ca

Alexander Cai  
acai2@ualberta.ca

An Zhang  
anzhang@hdu.edu.cn

<sup>1</sup> School of Mathematics and Statistics, Ningbo University, Ningbo, China

<sup>2</sup> Department of Computing Science, University of Alberta, Edmonton T6G 2E8, Alberta, Canada

<sup>3</sup> School of Economics and Management, Xi'an Technological University, Xi'an 710064, Shaanxi, China

<sup>4</sup> Department of Mathematics, Hangzhou Dianzi University, Hangzhou, China

pseudo-polynomial time dynamic programming exact algorithm and then, when the tardiness can be unbounded, we develop it into a fully polynomial time approximation scheme. The dynamic programming exact algorithm has a space complexity too high for truthful implementation; we propose an alternative to integrate the enumeration and the dynamic programming recurrences, followed by a depth-first-search walk in the reschedule space. We implemented the alternative exact algorithm in C and conducted numerical experiments to demonstrate its promising performance.

**Keywords** Scheduling · Rescheduling · Dynamic programming · Approximation scheme · Depth-first-search

## 1 Introduction

In many modern manufacturing production and service systems, it is not uncommon that a well-planned operating schedule is disrupted unexpectedly by the arrival of new jobs, job delays, job cancellations, and machine breakdowns. For example, the U.S. Department of Transportation reported that in November 2017 alone, 11.74% of their domestic flights were delayed and 0.3% were canceled; the major reasons causing these delays and cancellations include weather condition, airport flow control, short of aircraft and/or crew, and flight operating cost. Subsequently, rescheduling is required, to adjust the planned schedule to account for these disruptions without causing excessive changes.

Note that we typically use a “job” to refer to a task to be processed and a “machine” to refer to the system or a part of the system that processes the jobs; note also that by rescheduling we meant to compute a transient schedule that *matches up* with the planned schedule, but not a completely new scheduling discarding all prior planning efforts.

Wang et al. (2018) considered a variant of the multiprocessor rescheduling problem formulated out of an outpatient appointment making over a planning horizon. This is one of the first work known to us that combines two important sub-fields of studies, *scheduling with rejection* and *rescheduling*, both of which have a rich literature. Given a set of jobs to be processed on a number of identical machines to minimize the total job completion time, the decision-maker computes an optimal schedule by the shortest processing time (SPT) rule. The completion time for each job in the schedule is then communicated with and used as a guarantee for operational revenue. Before the schedule is actually executed, the decision-maker is told that some jobs could not arrive in time, but are delayed with their delayed arrival times for possible rescheduling. With the understanding that the completion time for a job in the original planned schedule is a guarantee, the completion time disruption for the job in the reschedule is expected bounded. However, possibly no feasible reschedule would exist, and in fact it is known NP-hard to determine such an existence (Hall and Potts 2010; Liu and Ro 2014; Zhao et al. 2016).

In Wang et al. (2018), the decision-maker can choose to reject some jobs, delayed or non-delayed, by paying the corresponding rejection cost, in order to minimize the total completion time of the accepted jobs, subject to a given upper bound on the

total rejection cost of the rejected jobs and a given upper bound on the completion time deviation of each accepted job in the reschedule. Wang et al. (2018) presented a mixed integer linear programming formulation, a pseudo-polynomial time dynamic programming exact algorithm and an exponential time branch-and-price approach.

In this paper, we study a rejection-allowed multiprocessor rescheduling problem with a similar setting as in Wang et al. (2018), but for a novel objective function to integrate the three cost components: the total completion time of the accepted jobs, the total rejection cost of the rejected jobs, and the maximum completion time disruption for the accepted jobs. Since the completion time for a job in the original planned schedule is communicated as a guarantee, we take it as the *virtual due date* for the job (Wang et al. 2017; Luo et al. 2020), and we measure the completion time disruption for an accepted job to be its tardiness in the reschedule. That is, the maximum completion time disruption for the accepted jobs is their maximum tardiness. (The completion time disruption in the model by Wang et al. (2018) is the completion time deviation. One will see later that our algorithms for the above objective function can be extended for the total tardiness of the accepted jobs, or the maximum completion time deviation of the accepted jobs, or the total completion time deviation of the accepted jobs.)

An application that motivates our objective function is operating room scheduling and rescheduling introduced in Addis et al. (2016), where we are asked to assign a set of elective patients to a set of available operating room blocks. In this application, a medium-term patient assignment for every following week is generated to minimize the total waiting time. When applying the weekly solution, unpredictable patient unavailability at the operating room start time may disrupt the planned schedule. Such disruption requires to be addressed by the decision-maker from the surgery management department in the short-term patient assignment, through rescheduling under the upper bounded total waiting time and the upper bounded tardiness of the operations. To facilitate the rescheduling process, the decision-maker could select some patients and transfer them to other hospitals at the associated costs or penalties; and their aim is to minimize a combination of the three cost components. In general there are multiple operating rooms, the formulated novel rescheduling problem generalizes the one studied by Wang et al. (2018) and several classic NP-hard scheduling problems.

Rescheduling has received much attention in the last three decades, and there are many interesting theoretical models formulated out of various domains of applications, such as automobile industry (Bean et al. 1991), Space Shuttle ground processing coordination (Zweben et al. 1993), shipyard portal crane access control (Clausen et al. 2001), CrewSolver decision-support system (Yu et al. 2003), and deregulated power markets (Dahal et al. 2015), to name a few. The rescheduling models often build on the corresponding classical scheduling problems, addressing the disruption either as a constraint or integrated as a part of the objective function. For example, Wu et al. (1992) considered the shop rescheduling to minimize the makespan and the total start processing time deviation. Below we briefly review some most closely related work, though Wang et al. (2018) has done an excellent one. We note that most past rescheduling research is on a single machine, in response to different types of disruption: Unal et al. (1997), Hall and Potts (2004), Hall et al. (2007) and Zhao et al. (2016), and most recently Zhang et al. (2021) investigated the case where a new set or multiple new sets of jobs are inserted into a planned schedule. Hall and Potts (2010)

considered a variant to respond to job delays. Liu and Ro (2014), Liu et al. (2018) and Luo et al. (2018) studied another variant to respond to machine unavailability, which is extended to multiple parallel identical machines by Yin et al. (2016). Besides minimizing the makespan or the total (weighted) job completion time in the above models, the decision-maker is asked to adjust the schedule to address disruptions, but not supposed to reject any jobs.

Scheduling with rejection was first studied by Bartal et al. (2000), on multiprocessors to minimize the makespan of the accepted jobs and the total rejection cost of the rejected jobs. Several others studied the total (weighted) job completion time, in various scheduling environments (Cheng and Sun 2007; Li and Lu 2017; Manavizadeh et al. 2013; Ou and Zhong 2017; Shabtay 2014; Yin et al. 2016; Zhang et al. 2010). Nevertheless, these work do not involve any rescheduling to respond to disruption.

There aren't many works on rescheduling with rejection in the literature. Besides the above mentioned work by Wang et al. (2018) to minimize the total job completion time (while subject to the upper bounded total rejection cost and maximum completion time deviation), Wang et al. (2017) investigated a bi-objective rejection-allowed rescheduling on a single machine to respond to continuous arrivals of new jobs, where the job processing time is a linear function of its start processing time and its amount of allocated resource; the two objectives are the operational cost which includes the total job completion time, the total rejection cost and the resource consumption, and the disruption cost which is the total tardiness, where the job completion time in the original schedule is regarded as its implied due date. Luo et al. (2020) considered the rejection-allowed rescheduling on a single machine, in response to a set of delayed jobs; for the same objective as ours, they presented a pseudo-polynomial time dynamic programming exact algorithm and transferred it into a *fully polynomial time approximation scheme* (FPTAS) when the total rejection cost can be unbounded. Luo et al. left an open question on whether their algorithm design techniques can be extended to multiple identical machines (Luo et al. 2020).

Recall that for multiprocessor scheduling to minimize the total job completion time, an optimal schedule can be obtained by the shortest processing time (SPT) rule, that is, one first sorts the jobs in non-decreasing order of their processing times and then assigns them sequentially to the next available machine. We may assume without loss of generality that the jobs are given sorted and  $m$  is the number of identical machines. Then in the original planned schedule, the  $j$ -th job is assigned to the  $i$ -th machine, where  $i = (j - 1) \% m + 1$ . In response to the delayed jobs, the decision-maker adjusts the schedule to accept a subset of jobs, either delayed or non-delayed, for processing, in order to minimize the operational cost which includes the total job completion time of the accepted jobs, the total rejection cost of the rejected jobs, and the maximum *virtual* tardiness of the accepted jobs. For calculating the virtual tardiness of an accepted job, its completion time in the planned schedule is taken as its virtual due date. We respect any given upper bound on the total rejection cost of the rejected jobs and any given upper bound on the tardiness of the accepted jobs, if any. Therefore, our rescheduling model differs from the works of Hall and Potts (2010) and Wang et al. (2018) since it does not penalize job earlier completion, and extends them by allowing the decision-maker to reject some jobs to reduce the negative effect of disruption. We remark that if the jobs have their given *actual* due dates, then computing a planned optimal

schedule that minimizes the maximum tardiness among the set of schedules achieving the minimum total completion time is already NP-hard (Gupta et al. 2003). In such a case, the disruption to the planned schedule measured as the maximum tardiness of the accepted jobs in the adjusted schedule can be difficult to justify.

We organize the rest of the paper as follows. In Sect. 2, we formally define our problem and the notations to be used throughout the paper. Section 3 is devoted to several important structural properties of the optimal schedules on the accepted jobs. Using these properties, we first present a pseudo-polynomial time dynamic programming exact algorithm for our multiprocessor rescheduling problem in Sect. 4, and then develop it into an FPTAS when the tardiness can be unbounded. In Sect. 5 we first discuss the challenging issues in implementing the dynamic programming exact algorithm, then propose an alternative to integrate the enumeration on the sets of accepted jobs and the dynamic programming recurrences; we also present the implementation details and the numerical experiments to demonstrate the promising performance of the alternative exact algorithm, with discussions. We conclude the paper in Sect. 6, and suggest some future research.

## 2 Problem definition

In this section, we formally define the rejection-allowed multiprocessor rescheduling problem to respond to job delays, and our objective function. We remind the readers that multiprocessor scheduling to minimize the total job completion time is solvable in polynomial time by the SPT rule (in contrary to minimizing the makespan, which is NP-hard Garey and Johnson 1979).

Let  $\mathcal{J} = \{1, 2, \dots, n\}$  denote the set of jobs each to be processed non-preemptively on one of the  $m$  parallel identical machines, and let  $p_j$  and  $e_j$  denote the processing time and the rejection cost of job  $j$ , respectively, for  $j = 1, 2, \dots, n$ . We assume  $p_j$  and  $e_j$  are non-negative integers. All these jobs are planned for processing in a schedule  $\pi^*$  to minimize the total completion time, and they are assumed, though not with 100% certainty, available at time point zero when the schedule starts. We note that such a schedule  $\pi^*$  can be constructed by the SPT rule in  $O(n \log n)$  time. To ease the presentation, we assume the jobs are given in the SPT order with an overhead of  $O(n \log n)$  time, that is,

$$p_1 \leq p_2 \leq \dots \leq p_n, \quad (1)$$

and thus in  $\pi^*$  job  $j$  is processed on machine  $i$  where  $i = (j - 1) \% m + 1$ . We also assume that all the jobs will be processed as early as possible in  $\pi^*$ , and their completion times satisfy

$$C_1(\pi^*) \leq C_2(\pi^*) \leq \dots \leq C_n(\pi^*). \quad (2)$$

These completion times  $C_j(\pi^*)$ 's are taken as the virtual due dates for the jobs, respectively.

Before executing the schedule  $\pi^*$ ,<sup>1</sup> the decision-maker is told that a set  $\mathcal{D} \subseteq \mathcal{J}$  of jobs are delayed to some later time point  $r_{\mathcal{D}} > 0$ , and thus might not be ready at their planned start processing times in  $\pi^*$ . The decision-maker needs to address such disruption, through rescheduling to minimize the overall operational cost, yet to achieve an acceptable service level. For reducing the negative impact of disruption, the decision-maker is allowed to reject some delayed or non-delayed<sup>2</sup> jobs from the rescheduling, by paying the rejection cost for each rejected job that represents the outsourcing cost or the customer goodwill. Nevertheless, a typical service level does not allow to reject jobs unlimitedly, but keeps a bound  $h$  on the total rejection cost; that is, if  $\mathcal{R}$  is the set of rejected jobs, then it should satisfy

$$\sum_{j \in \mathcal{R}} e_j \leq h. \quad (3)$$

Let  $\mathcal{A} = \mathcal{J} \setminus \mathcal{R}$ , which is the set of jobs accepted into rescheduling, and let  $\sigma$  denote a feasible reschedule for  $\mathcal{A}$  by the decision-maker. We define the following quantities for each accepted job  $j$ :

- $C_j(\pi^*)$ : the completion time (which becomes the virtual due date) of job  $j$  in the original planned schedule  $\pi^*$ ;
- $C_j(\sigma)$ : the completion time of job  $j$  in the reschedule  $\sigma$ ;
- $T_j(\sigma) = \max\{C_j(\sigma) - C_j(\pi^*), 0\}$ : the virtual tardiness of job  $j$  in  $\sigma$ .

We simplify  $C_j(\sigma)$  and  $T_j(\sigma)$  to  $C_j$  and  $T_j$ , respectively, when the reschedule  $\sigma$  is clear from the context.

Let  $T_{\max} = \max_{j \in \mathcal{A}} T_j$  denote the maximum tardiness of the accepted jobs in  $\sigma$ , which is bounded by a given threshold  $k$  per service level standard. That is, the feasibility of the reschedule  $\sigma$  states that

$$T_{\max} = \max_{j \in \mathcal{A}} T_j \leq k. \quad (4)$$

Besides the above two strict upper bounds in Eqs. (3, 4) on the total rejection cost and the maximum tardiness, respectively, the decision-maker aims to minimize their the total operational cost for rescheduling, consisting of three components: the total job completion time, the total rejection cost, and the maximum tardiness. To wrap them into a single objective, we set up a non-negative constant scaling factor  $\mu$  for the maximum tardiness (if needed, scaling the total rejection cost can be done through scaling individual job rejection cost).

We write our rescheduling problem in the three-field notation  $\alpha \mid \beta \mid \gamma$  (Graham et al. 1979), in which  $\alpha$  is the scheduling environment and  $\alpha = Pm$  indicates a constant number  $m$  of parallel identical machines;  $\beta$  describes the job characteristics and we use “rej” for rejection-allowed, “ $r_{\mathcal{D}}$ ” for a subset  $\mathcal{D}$  of jobs delayed to time point  $r_{\mathcal{D}}$ ,

<sup>1</sup> A standard treatment can be applied if the disruption is informed after the job processing has started, see Wang et al. (2018).

<sup>2</sup> We remark that in our general setting a non-delayed job can be rejected for cost recovery purpose, typically when its rejection cost is much lower compared to its completion time if it were kept for processing.

“ $\sum_{j \in \mathcal{R}} e_j \leq h$ ” for the total rejection cost being bounded by  $h$ , and “ $T_{\max} \leq k$ ” for the maximum tardiness being bounded by  $k$ ; and  $\gamma$  defines the objective functions to be minimized and in our case it is a single objective  $\sum_{j \in \mathcal{A}} C_j + \sum_{j \in \mathcal{R}} e_j + \mu T_{\max}$ . That is, our multiprocessor rescheduling problem is

$$Pm \mid \text{rej}, r_{\mathcal{D}}, \sum_{j \in \mathcal{R}} e_j \leq h, T_{\max} \leq k \mid \sum_{j \in \mathcal{A}} C_j + \sum_{j \in \mathcal{R}} e_j + \mu T_{\max}. \quad (5)$$

The above problem is NP-hard, since the special and classical case  $1 \mid r_{\mathcal{D}} \mid \sum C_j$ , that is, single machine scheduling with a nonzero release date, is already binary/weakly NP-hard (Rinnooy Kan 1976) (which can be obtained by setting  $m = 1$ ,  $e_j = 1$  for all  $j \in \mathcal{J}$  and  $h = 0$ ,  $k = +\infty$  and  $\mu = 0$  in (5)).

### 3 Structural properties

In this section, we prove a number of structural properties associated with the optimal reschedules for the multiprocessor rescheduling problem (5). These properties are then taken advantage to design the recurrences between feasible partial reschedules in the next section, which are used to develop a dynamic programming exact algorithm for computing an optimal reschedule.

Recall that without loss of generality the jobs are given in the SPT order (in Eq. (1)) and in the planned schedule  $\pi^*$  the job completion times or their due dates satisfy Eq. (2). In the sequel we refer a job by its index in the SPT order.

Assume  $\sigma^*$  is an optimal reschedule by the decision-maker in which the set of accepted jobs is  $\mathcal{A}$ . We remind the reader that a rejected job can either be delayed or non-delayed, at the decision-maker's wish. Using similar terminologies as in Wang et al. (2018) and Luo et al. (2020), we use the time point  $r_{\mathcal{D}}$  to partition the reschedule  $\sigma^*$  into the *earlier* and the *later* schedules. Namely, the earlier schedule is the prefix of  $\sigma^*$  consisting of the jobs that are started strictly before the time point  $r_{\mathcal{D}}$ ; the remainder of  $\sigma^*$  forms the later schedule. One sees that the delayed jobs can be accepted into the later schedule only. We also use the earlier (the later, respectively) schedule on the machine  $i$  to refer to the earlier (the later, respectively) sub-schedule of  $\sigma^*$  on the machine  $i$ . The next lemma summarizes the structural properties of  $\sigma^*$  which are useful for designing the recurrences.

**Lemma 1** *For Problem (5), there exists an optimal schedule  $\sigma^*$  for the accepted jobs in which:*

- (a) *all the jobs are processed as early as possible;*
- (b) *the jobs in the earlier schedule on each machine are processed in their SPT order;*
- (c) *the jobs in the later schedule on each machine are processed in their SPT order.*

**Proof** Note that once the accepted jobs have been decided, the total rejection cost is fixed, and for any job processing order, jobs are started at their earliest to minimize their respective completion times, which also minimize their possible tardiness. That is, item (a) holds for any optimal schedule for the accepted jobs.

To prove item (b), we notice that the jobs in the earlier schedule on a machine in  $\sigma^*$  could be processed by different machines in  $\pi^*$ , but they are all available at time point zero. Without loss of generality let us consider machine 1. Assume to the contrary that these jobs are not processed in their SPT order, and let  $j$  and  $i$  be a pair of adjacent jobs processed on machine 1 such that  $i < j$ , and let  $s$  be the start processing time of job  $j$ ; that is,  $C_j(\sigma^*) = s + p_j < r_{\mathcal{D}}$  and  $C_i(\sigma^*) = s + p_j + p_i$ . From  $p_i \leq p_j$ , if we swap these two jobs in the earlier schedule on machine 1 and denote the resultant schedule as  $\sigma'$ , then these two jobs are still in the earlier schedule on machine 1 with  $C_i(\sigma') = s + p_i < r_{\mathcal{D}}$  and  $C_j(\sigma') = s + p_i + p_j$ . Therefore,

$$\begin{aligned} C_i(\sigma') + C_j(\sigma') &= (s + p_i) + (s + p_i + p_j) \leq (s + p_j) + (s + p_i + p_j) \\ &= C_i(\sigma^*) + C_j(\sigma^*); \end{aligned}$$

that is, the total job completion time is not increased.

(We remark that the above proof shows  $p_i = p_j$ , for otherwise  $\sigma^*$  wouldn't be optimal; in this sense, if we relax the SPT order to allow jobs of the same processing time in an arbitrary sub-order, then all optimal reschedules have the properties in the lemma statement.)

On the other hand, by  $i < j$  and Eq. (2), the virtual due dates for these two jobs satisfy  $C_i(\pi^*) \leq C_j(\pi^*)$ . The maximum tardiness between them in  $\sigma^*$  is

$$T = \max\{s + p_j + p_i - C_i(\pi^*), s + p_j - C_j(\pi^*), 0\};$$

and the maximum tardiness between them in the new schedule  $\sigma'$  is

$$\begin{aligned} T' &= \max\{s + p_i - C_i(\pi^*), s + p_i + p_j - C_j(\pi^*), 0\} \\ &\leq \max\{s + p_i + p_j - C_i(\pi^*), 0\} \leq T. \end{aligned}$$

In other words, the maximum tardiness is not increased either. This proves item (b) that the jobs in the earlier schedule on each machine are processed in their SPT order.

Item (c) can be similarly proved using the fact that swapping two adjacent jobs in the later schedule on a machine keeps the two jobs in the later schedule, and we skip the details here.  $\square$

## 4 Algorithmic results

In this section, we first define the feasible partial reschedules to be computed, and then use the structural properties stated in Lemma 1 to design the recurrences between these feasible partial reschedules, and develop the recurrences into a dynamic programming exact algorithm for computing an optimal full reschedule to Problem (5). We show that such an algorithm runs in pseudo-polynomial time. Lastly, we transfer the dynamic programming exact algorithm into an FPTAS when the upper bound on the maximum tardiness is lifted, using the standard sparsing technique.



For every non-negative integer  $j$ , we denote  $[j] = \{0, 1, 2, \dots, j\}$ . We remark that in this notation, sometimes the element 0 does not have its actual correspondence but it serves as the symbolic boundary, that is,  $[j]$  might just refer to the set  $\{1, 2, 3, \dots, j\}$ .

#### 4.1 A dynamic programming exact algorithm

We define the feasible partial reschedules to be computed first, and then design recurrences for computing them based on the structural properties of the optimal reschedules stated in Lemma 1. At the end, we achieve a dynamic programming exact algorithm for constructing an optimal full reschedule.

A feasible partial reschedule is a combination of sub-schedules on the  $m$  machines, where a sub-schedule on a machine is defined by the total job processing time of the earlier schedule, the start processing time in the later schedule, and the total job processing time of the later schedule. The partial reschedule also includes the total job completion time, the maximum tardiness of the jobs accepted in the reschedule, and the total rejection cost for those jobs already been rejected. Note that the individual job information, such as whether it is accepted or rejected, and if it is accepted then on which machine it is processed etc., is not recorded in the partial reschedule but can be backtracked in the dynamic programming algorithm.

Recall that the jobs of  $\mathcal{J}$  are given in the SPT order with an overhead of  $O(n \log n)$  time, the original planned schedule  $\pi^*$  is computed and then the job completion times or their virtual dues  $C_j(\pi^*)$  are computed in  $O(n)$  time. Consider the job subset  $[j]$  containing the first  $j$  jobs in the SPT order, for any  $j = 0, 1, 2, \dots, j$ ;<sup>3</sup> we examine the *feasible partial reschedules* to process some jobs of  $[j]$ , satisfying the structural properties stated in Lemma 1.

Given a combination of the start processing times in the later schedules on the  $m$  machines, namely  $(s_1, s_2, \dots, s_m)$  such that  $s_1 \geq s_2 \geq \dots \geq s_m \geq r_{\mathcal{D}}$  (since these  $m$  machines are identical), we use a  $(2m + 4)$ -dimensional vector  $(j; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R)$  to represent a state, which is *true* if and only if there is an *associated* partial schedule on the job subset  $[j]$  such that

- (i) the total processing time in the earlier schedule on machine  $i$  is exactly  $M_i$ , for every  $i \in [m]$ ;
- (ii) the total processing time in the later schedule on machine  $i$  is exactly  $P_i$ , for every  $i \in [m]$ ;
- (iii) the total job completion time is exactly  $Z$ ;
- (iv) the maximum tardiness is exactly  $T$  and  $T \leq k$ ;
- (v) the total rejection cost is exactly  $R$  and  $R \leq h$ ;
- (vi) and for every  $i \in [m]$ ,  $M_i \leq s_i$ , and if  $j = n$  then  $\max\{M_i, r_{\mathcal{D}}\} = s_i$ .

The following recurrences in the dynamic programming algorithm compute from true states on  $[j]$  to true states on  $[j + 1]$ , sequentially for  $j = 0, 1, \dots, n - 1$ . Let  $p_{\max} = \max_{j \in \mathcal{J}} p_j$ . Then we only need to examine those  $s_i$ 's such that

$$r_{\mathcal{D}} - 1 + p_{\max} \geq s_1 \geq s_2 \geq \dots \geq s_m \geq r_{\mathcal{D}}, \quad (6)$$

<sup>3</sup> Here 0 serves as the boundary condition; for example,  $[0]$  represents the empty set.

and there are  $O(p_{\max}^m)$  such combinations. In the sequel we fix a combination  $(s_1, s_2, \dots, s_m)$  satisfying Eq. (6) for the discussion.

Clearly, when  $j = 0$ , the only true *initial* states are  $(0; 0, 0, \dots, 0, 0, 0, \dots, 0, 0, 0, 0)$ .

We show below that, in general, a true state  $(j; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R)$  with  $0 \leq j < n$  leads to no more than  $2m + 1$  true states of the form  $(j + 1; M'_1, M'_2, \dots, M'_m, P'_1, P'_2, \dots, P'_m, Z', T', R')$ , by extending an associated partial schedule to either reject job  $j + 1$ , or accept job  $j + 1$  into one of the earlier schedules, or accept job  $j + 1$  into one of the later schedules.

To determine precisely these true states of the form  $(j + 1; M'_1, M'_2, \dots, M'_m, P'_1, P'_2, \dots, P'_m, Z', T', R')$ , we distinguish the following two cases on job  $j + 1$  being delayed or not:

*Case 1* Job  $j + 1$  is non-delayed (i.e.,  $j + 1 \in \mathcal{J} \setminus \mathcal{D}$ ).

In this case, job  $j + 1$  can be rejected if  $R + e_{j+1} \leq h$  (*Comment*: the rejection budget allows so). On the other hand, if job  $j + 1$  is accepted, then it has to be appended to the end of an earlier schedule or a later schedule due to the SPT order. Appending to the earlier schedule on machine  $i$  requires  $M_i < r_{\mathcal{D}}$  (*Comment*: the earlier schedule can be extended),  $M_i + p_{j+1} \leq s_i$  (*Comment*: the start processing time of the later schedule is not violated), and its tardiness  $M_i + p_{j+1} - C_{j+1}(\pi^*) \leq k$  (*Comment*: the tardiness bound is respected); appending to the later schedule on machine  $i$  only requires its tardiness  $s_i + P_i + p_{j+1} - C_{j+1}(\pi^*) \leq k$ . In summary, there are at most  $2m + 1$  true states:

- $(j + 1; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R + e_{j+1})$ , if  $R + e_{j+1} \leq h$ ;
- $(j + 1; M_1, \dots, M_{i-1}, M_i + p_{j+1}, M_{i+1}, \dots, M_m, P_1, P_2, \dots, P_m, Z + (M_i + p_{j+1}), \max\{T, M_i + p_{j+1} - C_{j+1}(\pi^*)\}, R)$ , if  $M_i < r_{\mathcal{D}}$ ,  $M_i + p_{j+1} \leq s_i$ , and  $M_i + p_{j+1} - C_{j+1}(\pi^*) \leq k$ , where  $i \in [m]$ ;
- $(j + 1; M_1, M_2, \dots, M_m, P_1, \dots, P_{i-1}, P_i + p_{j+1}, P_{i+1}, \dots, P_m, Z + (s_i + P_i + p_{j+1}), \max\{T, s_i + P_i + p_{j+1} - C_{j+1}(\pi^*)\}, R)$ , if  $s_i + P_i + p_{j+1} - C_{j+1}(\pi^*) \leq k$ , where  $i \in [m]$ .

*Case 2* Job  $j + 1$  is delayed (i.e.,  $j + 1 \in \mathcal{D}$ ).

Similar to Case 1, in this case, job  $j + 1$  can be rejected if  $R + e_{j+1} \leq h$ . On the other hand, if job  $j + 1$  is accepted, then it has to be appended to the end of a later schedule due to the SPT order. Appending to the later schedule on machine  $i$  requires its tardiness  $s_i + P_i + p_{j+1} - C_{j+1}(\pi^*) \leq k$ . In summary, there are at most  $m + 1$  true states:

- $(j + 1; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R + e_{j+1})$ , if  $R + e_{j+1} \leq h$ ;
- $(j + 1; M_1, M_2, \dots, M_m, P_1, \dots, P_{i-1}, P_i + p_{j+1}, P_{i+1}, \dots, P_m, Z + (s_i + P_i + p_{j+1}), \max\{T, s_i + P_i + p_{j+1} - C_{j+1}(\pi^*)\}, R)$ , if  $s_i + P_i + p_{j+1} - C_{j+1}(\pi^*) \leq k$ , where  $i \in [m]$ .

At the end, we need to double check each (tentatively) true state of the form  $(n; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R)$ , and keep it true only if  $s_i = \max\{M_i, r_{\mathcal{D}}\}$  for every  $i \in [m]$ . Then, we calculate its objective value as  $Z + R + \mu T$ . By a standard backtracking from a state achieving the minimum objective value, one can obtain an optimal reschedule with the structural properties described in Lemma 1,

and respecting the two given bounds on the total rejection cost and the maximum tardiness, respectively.

We next bound the number of states used by the dynamic programming algorithm. Given two true states with the same values in the first  $2m + 1$  positions, that is,  $(j; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R)$  and  $(j; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z', T', R')$ , one clearly sees that the former is superior than the latter when  $Z \leq Z', T \leq T'$  and  $R \leq R'$ , because the former achieves a smaller objective value. Subsequently, we can choose to always use the superior true states in the dynamic programming computation. In other words, among these three  $Z$ -,  $T$ -,  $R$ -dimensions, we can choose to opt out an arbitrary one by keeping only the smallest value across all the true states.

The ranges for all the  $2m + 4$  dimensions are as follows:

$$\begin{aligned} 0 &\leq j \leq n, \\ 0 &\leq M_i \leq s_i \leq r_{\mathcal{D}} - 1 + p_{\max}, \quad i \in [m], \\ 0 &\leq P_i \leq \sum_{j \in \mathcal{J}} p_j \leq np_{\max}, \quad i \in [m], \\ 0 &\leq Z \leq n(r_{\mathcal{D}} + np_{\max}) \leq 2n^2 p_{\max}, \\ 0 &\leq T \leq k, \\ 0 &\leq R \leq \min \left\{ \sum_{j \in \mathcal{J}} e_j, h \right\} \leq h, \end{aligned} \quad (7)$$

where we assume without loss of generality that  $r_{\mathcal{D}} \leq np_{\max}$  and  $h \leq \sum_{j \in \mathcal{J}} e_j$ . We may also assume without loss of generality that  $k < np_{\max} (< 2n^2 p_{\max})$ , and choose to opt out the longer one of the  $Z$ - and  $R$ -dimensions. Therefore, the number of states in the tabular computation is at most  $n^{m+1} (r_{\mathcal{D}} + p_{\max})^m p_{\max}^{2m} k \min\{2n^2 p_{\max}, h\}$ , which is pseudo-polynomial in the size of input instance. Since each true state directly leads to at most  $2m + 1$  other true states, we have the following theorem.

**Theorem 1** *Problem (5) admits an  $O((2m+1)n^{m+1}(r_{\mathcal{D}}+p_{\max})^m p_{\max}^{2m} k \min\{2n^2 p_{\max}, h\})$ -time dynamic programming exact algorithm.*

## 4.2 An illustrating example

In this section, we use a small two-machine instance to illustrate the above dynamic programming exact algorithm. The instance contains ten jobs sorted in the SPT order, described in Table 1,  $\mathcal{D} = \{4, 5, 6, 8, 10\}$  and  $r_{\mathcal{D}} = 50$ , two bounds  $h = 100$  and  $k = 15$ , and the scaling factor  $\mu = 3$ . One thus easily calculates the job virtual due dates  $C_j(\pi^*)$ 's, included in Table 1.

From the allowed maximum tardiness  $T_{\max} \leq k = 15$  and  $r_{\mathcal{D}} = 50$ , one sees that only the jobs 8, 9, 10 can be processed in the later schedules. This fact can be effectively used to eliminate the state checking in the dynamic programming exact algorithm; for example, the delayed jobs 4, 5, 6 have to be rejected, resulting in a

**Table 1** The ten jobs in the two-machine instance, sorted in the SPT order

The five jobs assigned to machine 1 in $\pi^*$					The five jobs assigned to machine 2 in $\pi^*$				
job $j$	$p_j$	$e_j$	$C_j(\pi^*)$	Annotation	job $j$	$p_j$	$e_j$	$C_j(\pi^*)$	Annotation
1	2	12	2	p.E	2	5	60	5	E
3	7	40	9	E	4	11	20	16	R
5	18	10	27	R	6	21	38	37	R
7	28	44	55	E	8	30	25	67	p.L
9	35	40	90	E/L	10	42	61	109	L

In the last column “annotation”, an ‘E’ (‘L’, respectively) indicates the job has to be accepted into earlier (later, respectively) schedules; a ‘p.E’ (‘p.L’, respectively) indicates the job can be accepted into earlier (later, respectively) schedules; an ‘R’ indicates the job has to be rejected

reject cost of  $20 + 10 + 38 = 68$ , and consequently the room for rejecting the other jobs is left with  $100 - 68 = 32$  and thus the jobs 2, 3, 7 (into earlier schedules), 9, and 10 (into later schedules) have to be accepted. We add these as annotations in Table 1 for running the dynamic programming algorithm (though one doesn’t have to do this).

We also notice that the range of the total completion time is significantly larger than  $h$ , and thus choose to opt out the  $Z$ -dimension in the dynamic programming computation, that is, an inferior true state is eliminated when a superior true state is identified.

Below we illustrate the execution for  $(s_1, s_2) = (r_{\mathcal{D}} + 13, r_{\mathcal{D}}) = (63, 50)$  only, to obtain an optimal reschedule under the constraint that the later schedules on machines 1 and 2 start at time points 63 and 50, respectively. The following Table 2 lists the first a few and the last a few transitions from a true state with  $j$  to a true state with  $j + 1$  (the complete list of transitions are in Table 4 in the appendix), for  $j = 0, 1, 2, 3, 6, 7, 8, 9, 10$ , in each of which we give every true state a number, and the third column records from which true state and in which case it is first time derived true, except the initial true state. The last column contains additional comments, if any, for possibly better implementations of exact algorithms (see Sect. 5), and the objective values for those true states  $(n; M_1, M_2, P_1, P_2, Z, T, R)$  with  $s_i = \max\{M_i, r_{\mathcal{D}}\}$  for each machine  $i$ .

There are six true states of the form  $(10; 63, \cdot, \cdot, \cdot, \cdot, \cdot)$  corresponding to six feasible reschedules under the constraint  $(s_1, s_2) = (63, 50)$ , respectively, and the minimum objective value is 314. One can trace back the optimal reschedule in which the earlier schedule on machine 1 processes the jobs 7, 9, the later schedule on machine 1 is empty, the earlier schedule on machine 2 processes the jobs 1, 2, 3, and the later schedule on machine 2 processes the job 10.

#### 4.3 A fully polynomial time approximation scheme

Recall that our multiprocessor rescheduling Problem (5) is NP-hard, since the special and classical case  $1 \mid r_{\mathcal{D}} \mid \sum C_j$ , that is, single machine scheduling with a nonzero

**Table 2** The execution of the dynamic programming exact algorithm for  $(s_1, s_2) = (63, 50)$ , listing all the transitions from  $j$  to  $j + 1$ , for  $j = 0, 1, 2, 3, 6, 7, 8, 9, 10$ , while using the job annotations from Table 1. In particular,  $j$  jumps from 3 to 6 since all the three jobs 4, 5, 6 are rejected

No.	True state	Source state	Comments/objective value
1	(0; 0, 0, 0, 0, 0, 0)		
2	(1; 0, 0, 0, 0, 0, 12)	1, Case 1	Job 1 can be accepted into earlier's
3	(1; 2, 0, 0, 0, 2, 0, 0)	1, Case 1	
4	(1; 0, 2, 0, 0, 2, 0, 0)	1, Case 1	
5	(2; 5, 0, 0, 0, 5, 0, 12)	2, Case 1	Job 2 is accepted into earlier's
6	(2; 0, 5, 0, 0, 5, 0, 12)	2, Case 1	
7	(2; 7, 0, 0, 0, 9, 2, 0)	3, Case 1	
8	(2; 2, 5, 0, 0, 7, 0, 0)	3, Case 1	Job 3 is accepted into earlier's
9	(2; 5, 2, 0, 0, 7, 0, 0)	4, Case 1	
10	(2; 0, 7, 0, 0, 9, 2, 0)	4, Case 1	
11	(3; 12, 0, 0, 0, 17, 3, 12)	5, Case 1	
12	(3; 5, 7, 0, 0, 12, 0, 12)	5, Case 1	
13	(3; 7, 5, 0, 0, 12, 0, 12)	6, Case 1	
.....			
168	(9; 5, 37, 35, 30, 231, 13, 68)	87, Case 1	
169	(9; 63, 14, 0, 0, 114, 5, 93)	88, Case 1	
170	(9; 28, 49, 0, 0, 100, 5, 93)	88, Case 1	
171	(9; 28, 14, 35, 0, 149, 5, 93)	88, Case 1	
172	(9; 28, 14, 0, 35, 131, 5, 93)	88, Case 1	
173	(9; 63, 14, 0, 30, 194, 13, 68)	89, Case 1	
174	(9; 28, 49, 0, 30, 180, 13, 68)	89, Case 1	
175	(9; 28, 14, 35, 30, 229, 13, 68)	89, Case 1	
176	(9; 35, 42, 0, 0, 100, 5, 93)	90, Case 1	
177	(9; 0, 42, 35, 0, 163, 5, 93)	90, Case 1	
178	(9; 0, 42, 0, 35, 145, 5, 93)	90, Case 1	
179	(9; 35, 42, 0, 30, 180, 13, 68)	91, Case 1	
180	(9; 0, 42, 35, 30, 243, 13, 68)	91, Case 1	
181	(10; 63, 12, 42, 30, 293, 13, 80)	104, Case 2	Job 10 is accepted into later's/412
182	(10; 63, 12, 0, 72, 310, 13, 80)	104, Case 2	
183	(10; 63, 14, 42, 0, 219, 5, 93)	169, Case 2	
184	(10; 63, 14, 0, 42, 206, 5, 93)	169, Case 2	
185	(10; 63, 14, 42, 30, 299, 13, 68)	173, Case 2	
186	(10; 63, 14, 0, 72, 316, 13, 68)	173, Case 2	

release date, is already binary/weakly NP-hard (Rinnooy Kan 1976) (by setting  $m = 1$ ,  $e_j = 1$  for all  $j \in \mathcal{J}$  and  $h = 0$ ,  $k = +\infty$  and  $\mu = 0$  in Problem (5)).

From Lemma 1 statement, the intractability of our Problem (5) comes from two places where one decides the set  $\mathcal{A}$  of accepted jobs (including both non-delayed and delayed) and where one decides how to assign them to the  $m$  earlier schedules and the  $m$

later schedules. These decisions determine the total rejection cost and the jobs assigned to each of the earlier schedules and the later schedules, and henceforth determine the start processing time of each later schedule and the maximum job tardiness  $T_{\max}$ .

The NP-hardness of  $1 \mid r_{\mathcal{D}} \mid \sum C_j$  proven in Rinnooy Kan (1976) is by a reduction from the KNAPSACK/SUBSET-SUM problem (Garey and Johnson 1979), which equivalently states that it is NP-complete to decide the existence of a schedule in which the later schedule starts exactly at time point  $r_{\mathcal{D}}$ . We highlight this fact in the following lemma.

**Lemma 2** Rinnooy Kan (1976) *For a special case of Problem (5) in which  $(m, h, k, \mu) = (1, 0, +\infty, 0)$ , it is NP-hard to determine whether or not there exists a reschedule in which the later schedule starts exactly at time point  $r_{\mathcal{D}}$ .*

In this section, we use the standard sparsing technique to transfer the pseudo-polynomial time dynamic programming exact algorithm in Sect. 4.1 into polynomial time approximation schemes. However, one can generalize Lemma 2 to show that it is NP-hard to determine whether or not there exists a reschedule associated with any fixed combination of start processing times (for later schedules); on the other hand, we cannot afford to enumerate all the, pseudo-polynomially many, possible combinations. We thus first sample only polynomially many combinations through the sparsing technique. Recall that in the exact algorithm, a combination of starting processing times for the later schedules is  $(s_1, s_2, \dots, s_m)$ , satisfying Eq. (6). In the following approximation scheme, however, we use a different collection  $\mathcal{S}$  of only polynomially many combinations.

Given a small positive  $\epsilon$ , let,

$$\delta = 1 + \frac{\epsilon}{2n}, \quad v_0 = n + \left\lceil \log_{\delta} \frac{r_{\mathcal{D}} - 1 + p_{\max}}{r_{\mathcal{D}}} \right\rceil, \quad \text{and } I^s = \{t_{\ell} = r_{\mathcal{D}}\delta^{\ell}, 0 \leq \ell \leq v_0\}. \quad (8)$$

Define the collection of combinations

$$\mathcal{S} = \{(s_1, s_2, \dots, s_m) \mid s_1 \geq s_2 \geq \dots \geq s_m, s_i \in I^s, 1 \leq i \leq m\}; \quad (9)$$

note that  $|\mathcal{S}| \leq (v_0 + 1)^m \in O(4^m n^m (\log M)^m / \epsilon^m)$ , where  $M = \max\{r_{\mathcal{D}}, p_{\max}\}$ . In the following approximation scheme, for a combination  $(s_1, s_2, \dots, s_m) \in \mathcal{S}$ , a state  $(j; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R)$  is true if and only if there is an associated partial schedule on the job subset  $[j]$  such that

- (i) the total processing time in the earlier schedule on machine  $i$  is exactly  $M_i$ , for every  $i \in [m]$ ;
- (ii) the total processing time in the later schedule on machine  $i$  is exactly  $P_i$ , for every  $i \in [m]$ ;
- (iii) the total job completion time is exactly  $Z$ ;
- (iv) the maximum tardiness is exactly  $T$ ; (*Comment: The constraint “ $T \leq k$ ” used in the exact algorithm is removed.*)
- (v) the total rejection cost is exactly  $R$  and  $R \leq h$ ;

- (vi) and for every  $i \in [m]$ ,  $M_i \leq s_i$ . (Comment: The constraint “if  $j = n$  then  $\max\{M_i, r_{\mathcal{D}}\} = s_i$ ” used in the exact algorithm is removed.)

Additionally, from  $r_{\mathcal{D}} \leq s_i \leq \delta^n(r_{\mathcal{D}} - 1 + p_{\max}) \leq (1 + \epsilon)(r_{\mathcal{D}} - 1 + p_{\max})$ , we could have some jobs in the earlier schedule on the machine  $i$  starting after the time point  $r_{\mathcal{D}}$  and thus these jobs could be moved to the later schedule on the machine  $i$  to further decrease the objective value. We nevertheless leave the partial schedule as it is (as we only aim at near-optimality).

**Lemma 3** *Given a combination of start processing times  $(s_1, s_2, \dots, s_m)$  for which a state  $(n; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R)$  is evaluated true by the exact algorithm, there exists a combination  $(s'_1, s'_2, \dots, s'_m) \in \mathcal{S}$  such that*

- (1)  $s_i \leq s'_i < \delta s_i$  for  $1 \leq i \leq m$ ,
- (2) and there is a true state  $(n; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z', T', R)$  with  $Z' < \delta Z$  and  $T' < T + (\delta - 1)(r_{\mathcal{D}} - 1 + p_{\max})$ .

**Proof** For each  $i$ ,  $r_{\mathcal{D}} \leq s_i \leq r_{\mathcal{D}} - 1 + p_{\max}$ ; therefore, let  $\ell \in [v_0]$  be the smallest index such that  $s_i \leq t_\ell$  and let  $s'_i = t_\ell$ . It follows that  $s_i \leq s'_i < \delta s_i$ .

For the combination  $(s'_1, s'_2, \dots, s'_m)$ , by accepting the same set of jobs and assigning the same subset of accepted jobs to each of the earlier and the later schedules as in a reschedule  $\sigma$  associated with the true state  $(n; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R)$ , one sees that we achieve a new reschedule  $\sigma'$  which differs from  $\sigma$  only in the start processing times of the later schedules.

Assume a job  $j$  is assigned in the later schedule of  $\sigma'$  on machine  $i$  and let  $C'_j$  denote its completion time; let  $C_j$  denote its completion time in  $\sigma$ . We have  $C'_j - C_j = s'_i - s_i < (\delta - 1)s_i < (\delta - 1)C_j$ . In other words, the extra contribution of job  $j$  to the total job completion time is less than  $(\delta - 1)C_j$  and to the maximum tardiness is less than  $(\delta - 1)s_i \leq (\delta - 1)(r_{\mathcal{D}} - 1 + p_{\max})$ . We thus conclude that there is a true state  $(n; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z', T', R)$  with  $Z' < \delta Z$  and  $T' < T + (\delta - 1)(r_{\mathcal{D}} - 1 + p_{\max})$ .  $\square$

For any given combination  $(s_1, s_2, \dots, s_m) \in \mathcal{S}$ , we deploy almost the same recurrences in the exact dynamic programming algorithm to “sparsely” search for a near optimal reschedule, by Lemma 3 to relax the maximum tardiness to be unbounded (i.e.,  $k \geq s_1 + np_{\max}$ ). We note that, nevertheless, the maximum tardiness is still kept as a component cost in the overall objective function. In such a sparsing technique, essentially we first partition each dimension of pseudo-polynomial range into intervals of geometrically increasing lengths, resulting in only polynomially many intervals along that dimension; then for every grid in the  $(2m + 4)$ -dimensional space of states, we record at most one true state as its representative; we prove lastly that the propagated objective value increase from the initial true state to the final output reschedule can be controlled within any given fraction  $\epsilon > 0$ .

Recall that a state is represented as a  $(2m + 4)$ -dimensional vector  $(j; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R)$ , where the range of each variable is stated in Eq. (7). We thus see that all but the first dimensions have pseudo-polynomial ranges, and they will be partitioned. For the given fraction  $\epsilon > 0$ , denote the following algorithm as  $A(\epsilon)$ , which is a dynamic programming and has three steps:

In the first step, denote the following dimension ranges as

$$S_1 = np_{\max}, S_2 = n(s_1 + np_{\max}), S_3 = s_1 + np_{\max}, S_4 = h, \quad (10)$$

and let

$$v_\ell = \begin{cases} \lceil \log_\delta s_\ell \rceil, & \text{if } \ell = 1, 2, \dots, m; \\ \lceil \log_\delta S_1 \rceil, & \text{if } \ell = m + 1, m + 2, \dots, 2m; \\ \lceil \log_\delta S_2 \rceil, & \text{if } \ell = 2m + 1; \\ \lceil \log_\delta S_3 \rceil, & \text{if } \ell = 2m + 2; \\ \lceil \log_\delta S_4 \rceil, & \text{if } \ell = 2m + 3; \end{cases} \quad (11)$$

then the  $\ell$ -th dimension is partitioned into  $v_\ell + 1$  intervals of geometrically increasing lengths with ratio  $\delta$ , as follows:

$$\begin{aligned} I_0^\ell &= [0, 1); I_i^\ell = [\delta^{i-1}, \delta^i), 1 \leq i \leq v_\ell - 1; I_{v_\ell}^\ell \\ &= \begin{cases} [\delta^{v_\ell-1}, s_\ell], & \text{if } \ell = 1, 2, \dots, m; \\ [\delta^{v_\ell-1}, S_1], & \text{if } \ell = m + 1, m + 2, \dots, 2m; \\ [\delta^{v_\ell-1}, S_2], & \text{if } \ell = 2m + 1; \\ [\delta^{v_\ell-1}, S_3], & \text{if } \ell = 2m + 2; \\ [\delta^{v_\ell-1}, S_4], & \text{if } \ell = 2m + 3. \end{cases} \end{aligned} \quad (12)$$

In the sparsing scheme, for any given  $(2m + 3)$ -dimensional vector

$$(i_\ell, 1 \leq \ell \leq 2m + 3) \in \Pi_{\ell=1}^{2m+3}[v_\ell],$$

we define a grid  $G(j; i_\ell, 1 \leq \ell \leq 2m + 3)$  to be the collection of all the states  $(j; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R)$ , such that  $M_\ell \in I_{i_\ell}^\ell$  for  $1 \leq \ell \leq m$ ,  $P_\ell \in I_{i_{m+\ell}}^{m+\ell}$  for  $1 \leq \ell \leq m$ ,  $Z \in I_{i_{2m+1}}^{2m+1}$ ,  $T \in I_{i_{2m+2}}^{2m+2}$  and  $R \in I_{i_{2m+3}}^{2m+3}$ . The truth value of a grid is the disjunction of the truth values of all its member states; furthermore, when a grid is true, one of its true member state is selected as the *representative* of the grid. Note that the initial state  $(0; 0, 0, \dots, 0)$  is true, so is the grid  $G(0; 0, 0, \dots, 0)$ . The initial state  $(0; 0, 0, \dots, 0)$  is picked as the representative for the grid  $G(0; 0, 0, \dots, 0)$ .

In the second step of the FPTAS, the main task is to determine which grids are true and for each true grid to select its representative. This is done in the similar fashion as in the dynamic programming exact algorithm through recurrences, to be detailed below.

Given a general true grid  $G(j; i_\ell, 1 \leq \ell \leq 2m + 3)$ , let  $(j; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R)$  be its representative, which is a true state. From the argument in the exact algorithm, we know that this true state leads to up to  $2m + 1$  true states of the form  $(j + 1; M'_1, M'_2, \dots, M'_m, P'_1, P'_2, \dots, P'_m, Z', T', R')$  and thus they enable up to  $2m + 1$  grids of the form  $G(j + 1; i'_\ell, 1 \leq \ell \leq 2m + 3)$  to be true. We always pick the true state with the minimum total rejection cost as the representative for a true grid  $G(j + 1; i'_\ell, 1 \leq \ell \leq 2m + 3)$ . In more details, when a true state  $(j + 1; M'_1, M'_2, \dots, M'_m, P'_1, P'_2, \dots, P'_m, Z', T', R')$  is newly identified and



it belongs to the grid  $G(j+1; i'_\ell, 1 \leq \ell \leq 2m+3)$ , if the grid was not previously identified as true, then it becomes true and the true state is picked as its representative; if the grid had been previously identified as true with its representative  $(j+1; M''_1, M''_2, \dots, M''_m, P''_1, P''_2, \dots, P''_m, Z'', T'', R'')$  such that  $R'' \leq R'$ , then the representative remains unchanged; if the grid had been previously identified as true with its representative  $(j+1; M''_1, M''_2, \dots, M''_m, P''_1, P''_2, \dots, P''_m, Z'', T'', R'')$  such that  $R'' > R'$ , then the state  $(j+1; M'_1, M'_2, \dots, M'_m, P'_1, P'_2, \dots, P'_m, Z', T', R')$  replaces to be the representative. As side information, the source grid *leading to* the grid  $G(j+1; i'_\ell, 1 \leq \ell \leq 2m+3)$  needs to be updated properly, to be used in the backtracking process.

In the last step, the algorithm examines all true grids of the form  $G(n; i_\ell, 1 \leq \ell \leq 2m+3)$ , and for each calculates the objective value of its representative state  $(n; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R)$  as  $Z + R + \mu T$ . The same as in the exact algorithm, by a standard backtracking from a state achieving the minimum objective value, denoted as  $(n; M_1^\epsilon, M_2^\epsilon, \dots, M_m^\epsilon, P_1^\epsilon, P_2^\epsilon, \dots, P_m^\epsilon, Z^\epsilon, T^\epsilon, R^\epsilon)$ , one can obtain a full reschedule with the structural properties described in Lemma 1, respecting the given bound on the total rejection cost.

We remark that though the maximum tardiness is unbounded, one may adjust the scaling factor  $\mu$  to penalize a large maximum tardiness. We also remark that one can apply a similar reasoning to opt out one of the three  $Z$ -,  $T$ - and  $R$ -dimensions in the dynamic programming computation, but probably too costly to search through all the representatives saved for the related grids. The following theorem summarizes the approximability result for Problem (5).

**Theorem 2** *Given an  $\epsilon > 0$ , the algorithm  $A(\epsilon)$  is an  $O((2m+1)2^{6m+6}n^{3m+4}(\log M)^{3m+3}/\epsilon^{3m+3})$ -time  $(1+\epsilon)$ -approximation algorithm for Problem (5) with the unbounded maximum tardiness, where  $M = \max\{n, r_D, p_{\max}, h\}$ .*

**Proof** We first prove the performance ratio of the algorithm  $A(\epsilon)$ . We in fact want to prove that, given a combination  $(s_1, s_2, \dots, s_m)$  of start processing times and a true state  $(j; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R)$  in the exact algorithm, the algorithm  $A(\epsilon)$  picks a state  $(j; M'_1, M'_2, \dots, M'_m, P'_1, P'_2, \dots, P'_m, Z', T', R')$  to be a representative for some combination  $(s'_1, s'_2, \dots, s'_m) \in \mathcal{S}$ , satisfying  $s'_\ell \leq \delta^j s_\ell$  for  $1 \leq \ell \leq m$ ,  $M'_\ell \leq \delta^j M_\ell$  for  $1 \leq \ell \leq m$ ,  $P'_\ell \leq \delta^j P_\ell$  for  $1 \leq \ell \leq m$ ,  $Z' \leq \delta^j Z$ ,  $T' \leq \delta^j T + (\delta^j - 1)Z$ , and  $R' \leq R$ . The proof is done by an induction on  $j$ .

The statement holds trivially when  $j = 0$ , because there is only one true state  $(0; 0, \dots, 0)$  for any combination  $(s_1, s_2, \dots, s_m)$  in the exact algorithm and in the algorithm  $A(\epsilon)$ . For the combination  $(s'_1, s'_2, \dots, s'_m) \in \mathcal{S}$ , where  $s'_\ell$  is the largest value in  $I^s$  that is less than or equal to  $s_\ell$  for each  $\ell = 1, 2, \dots, m$ , this true state is picked as a representative by the algorithm  $A(\epsilon)$  for the initial grid  $G(0; 0, \dots, 0)$ .

Assume the statement holds for  $j$ , where  $0 \leq j \leq n-1$ , and we proceed to consider job  $j+1$ . That is, suppose  $(j; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R)$  is a true state for the combination  $(s_1, s_2, \dots, s_m)$  in the exact algorithm, then the algorithm  $A(\epsilon)$  has picked a true state  $(j; M'_1, M'_2, \dots, M'_m, P'_1, P'_2, \dots, P'_m, Z', T', R')$

for some combination  $(s'_1, s'_2, \dots, s'_m) \in \mathcal{S}$  as a representative, satisfying

$$\begin{cases} s'_\ell \leq \delta^j s_\ell, \text{ for } 1 \leq \ell \leq m, \\ M'_\ell \leq \delta^j M_\ell, \text{ for } 1 \leq \ell \leq m, \\ P'_\ell \leq \delta^j P_\ell, \text{ for } 1 \leq \ell \leq m, \\ Z' \leq \delta^j Z, \\ T' \leq \delta^j T + (\delta^j - 1)Z, \\ R' \leq R. \end{cases}$$

In the exact algorithm, the true state  $(j; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R)$  leads to up to  $2m + 1$  true states of the form  $(j + 1; \cdot, \cdot, \dots, \cdot)$ , distinguished in three cases.

If job  $j + 1$  can be rejected, that is,  $R + e_{j+1} \leq h$ , then the state  $(j + 1; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R + e_{j+1})$  is true for the combination  $(s_1, s_2, \dots, s_m)$ . Since  $R' \leq R$ , one sees that the state  $(j + 1; M'_1, M'_2, \dots, M'_m, P'_1, P'_2, \dots, P'_m, Z', T', R' + e_{j+1})$  is evaluated by the algorithm  $A(\epsilon)$  to be true for the combination  $(s'_1, s'_2, \dots, s'_m) \in \mathcal{S}$ . If  $(j + 1; M'_1, M'_2, \dots, M'_m, P'_1, P'_2, \dots, P'_m, Z', T', R' + e_{j+1})$  is picked as a representative, then we have proved the statement for the state  $(j + 1; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R + e_{j+1})$ . Otherwise, we conclude that the grid, to which  $(j + 1; M'_1, M'_2, \dots, M'_m, P'_1, P'_2, \dots, P'_m, Z', T', R' + e_{j+1})$  belongs, has a representative for the combination  $(s'_1, s'_2, \dots, s'_m)$  denoted as  $(j + 1; M''_1, M''_2, \dots, M''_m, P''_1, P''_2, \dots, P''_m, Z'', T'', R'')$ , such that  $M''_\ell \leq \delta M'_\ell$ ,  $P''_\ell \leq \delta P'_\ell$ , for  $1 \leq \ell \leq m$ ,  $Z'' \leq \delta Z'$ ,  $T'' \leq \delta T'$ , and  $R'' \leq R' + e_{j+1}$ , since the maximum ratio between two values inside the same interval is capped by  $\delta$ ; therefore,

$$\begin{cases} s'_\ell \leq \delta^j s_\ell, \\ M''_\ell \leq \delta M'_\ell, \\ P''_\ell \leq \delta P'_\ell, \\ Z'' \leq \delta Z', \\ T'' \leq \delta T', \\ R'' \leq R' + e_{j+1}, \end{cases} + \begin{cases} M'_\ell \leq \delta^j M_\ell, \\ P'_\ell \leq \delta^j P_\ell, \\ Z' \leq \delta^j Z, \\ T' \leq \delta^j T + (\delta^j - 1)Z, \\ R' \leq R, \end{cases} \implies \begin{cases} s'_\ell \leq \delta^{j+1} s_\ell, \\ M''_\ell \leq \delta^{j+1} M_\ell, \\ P''_\ell \leq \delta^{j+1} P_\ell, \\ Z'' \leq \delta^{j+1} Z, \\ T'' \leq \delta^{j+1} T + (\delta^{j+1} - 1)Z, \\ R'' \leq R + e_{j+1}. \end{cases}$$

i.e., the statement holds for the state  $(j + 1; M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R + e_{j+1})$  too.<sup>4</sup>

If job  $j + 1$  can be accepted into the earlier schedule on machine  $i$ , that is,  $M_i + p_{j+1} \leq s_i$ , then the state  $(j + 1; M_1, \dots, M_{i-1}, M_i + p_{j+1}, M_{i+1}, \dots, M_m, P_1, P_2, \dots, P_m, Z + M_i + p_{j+1}, \max\{T, M_i + p_{j+1} - C_{j+1}(\pi^*)\}, R)$  is true for the combination  $(s_1, s_2, \dots, s_m)$ . For the picked representative state  $(j; M'_1, M'_2, \dots, M'_m, P'_1, P'_2, \dots, P'_m, Z', T', R')$ , since  $M'_i + p_{j+1} < \delta^j (M_i + p_{j+1}) < \delta^j s_i$ , let  $s''_i$  be the smallest value in  $I^s$  that is greater than or equal to  $M'_i + p_{j+1}$ ; it follows that  $s''_i < \delta (M'_i + p_{j+1}) < \delta^{j+1} s_i$ . One sees that the state  $(j + 1; M'_1, \dots, M'_{i-1}, M'_i + p_{j+1}, M'_{i+1}, \dots, M'_m, P'_1, P'_2, \dots, P'_m, Z' + M'_i + p_{j+1}, \max\{T', M'_i + p_{j+1} - C_{j+1}(\pi^*)\}, R')$  is evaluated by the algorithm  $A(\epsilon)$  to be true for the combination  $(s'_1, \dots, s'_{i-1}, s''_i, s'_{i+1}, \dots, s'_m) \in \mathcal{S}$ .<sup>5</sup> For the true grid to which this state belongs, the

<sup>4</sup> In Case 1, no new combination of  $\mathcal{S}$  is involved.

<sup>5</sup> In Case 2, a possibly new combination  $(s'_1, \dots, s'_{i-1}, s''_i, s'_{i+1}, \dots, s'_m) \in \mathcal{S}$  is involved.

picked representative state is denoted as  $(j+1; M_1'', M_2'', \dots, M_m'', P_1'', P_2'', \dots, P_m'', Z'', T'', R'')$  and satisfies

$$\begin{cases} s_i'' < \delta^{j+1} s_i, \\ s_\ell'' < \delta^j s_\ell, \text{ if } \ell \neq i, \\ M_i'' \leq \delta(M_i' + p_{j+1}), \\ M_\ell'' \leq \delta M_\ell', \text{ if } \ell \neq i, \\ P_\ell'' \leq \delta P_\ell', \\ Z'' \leq \delta(Z' + M_i' + p_{j+1}), \\ T'' \leq \delta \max\{T', M_i' + p_{j+1} - C_{j+1}(\pi^*)\}, \\ R'' \leq R', \end{cases}$$

Using

$$\begin{aligned} M_i' + p_{j+1} &\leq \delta^j M_i + p_{j+1} < \delta^j (M_i + p_{j+1}), \\ Z' + M_i' + p_{j+1} &\leq \delta^j Z + \delta^j M_i + p_{j+1} < \delta^j (Z + M_i + p_{j+1}), \\ \max\{T', M_i' + p_{j+1} - C_{j+1}(\pi^*)\} \\ &\leq \max\{\delta^j T + (\delta^j - 1)Z, \delta^j M_i + p_{j+1} - C_{j+1}(\pi^*)\} \\ &\leq \begin{cases} \max\{\delta^j T + (\delta^j - 1)Z, \delta^j M_i - M_i\}, & \text{if } M_i + p_{j+1} \leq C_{j+1}(\pi^*), \\ \max\{\delta^j T + (\delta^j - 1)Z, \delta^j (M_i + p_{j+1} - C_{j+1}(\pi^*)) \\ \quad + (\delta^j - 1)(M_i + p_{j+1})\}, & \text{otherwise,} \end{cases} \\ &\leq \begin{cases} \delta^j T + (\delta^j - 1)(Z + M_i), & \text{if } M_i + p_{j+1} \leq C_{j+1}(\pi^*), \\ \delta^j \max\{T, M_i + p_{j+1} - C_{j+1}(\pi^*)\} \\ \quad + (\delta^j - 1)(Z + M_i + p_{j+1}), & \text{otherwise,} \end{cases} \\ &\leq \delta^j \max\{T, M_i + p_{j+1} - C_{j+1}(\pi^*)\} + (\delta^j - 1)(Z + M_i + p_{j+1}), \end{aligned}$$

we have

$$\begin{aligned} &\begin{cases} s_i'' < \delta^{j+1} s_i, \\ s_\ell'' < \delta^j s_\ell, \text{ if } \ell \neq i, \\ M_i'' \leq \delta(M_i' + p_{j+1}), \\ M_\ell'' \leq \delta M_\ell', \text{ if } \ell \neq i, \\ P_\ell'' \leq \delta P_\ell', \\ Z'' \leq \delta(Z' + M_i' + p_{j+1}), \\ T'' \leq \delta \max\{T', M_i' + p_{j+1} - C_{j+1}(\pi^*)\}, \\ R'' \leq R', \end{cases} + \begin{cases} M_\ell' \leq \delta^j M_\ell, \\ P_\ell' \leq \delta^j P_\ell, \\ Z' \leq \delta^j Z, \\ T' \leq \delta^j T + (\delta^j - 1)Z, \\ R' \leq R, \end{cases} \\ \Rightarrow &\begin{cases} s_i'' < \delta^{j+1} s_i, \\ s_\ell'' < \delta^j s_\ell, \text{ if } \ell \neq i, \\ M_i'' \leq \delta^{j+1} (M_i + p_{j+1}), \\ M_\ell'' \leq \delta^{j+1} M_\ell, \text{ if } \ell \neq i, \\ P_\ell'' \leq \delta^{j+1} P_\ell, \\ Z'' \leq \delta^{j+1} (Z + M_i + p_{j+1}), \\ T'' \leq \delta^{j+1} \max\{T, M_i + p_{j+1} - C_{j+1}(\pi^*)\} + (\delta^{j+1} - 1)(Z + M_i + p_{j+1}), \\ R'' \leq R; \end{cases} \end{aligned}$$

that is, the statement holds for the state  $(j+1; M_1, \dots, M_{i-1}, M_i + p_{j+1}, M_{i+1}, \dots, M_m, P_1, P_2, \dots, P_m, Z + M_i + p_{j+1}, \max\{T, M_i + p_{j+1} - C_{j+1}(\pi^*)\}, R)$ .

If job  $j+1$  can be accepted into the later schedule on machine  $i$ , then the state  $(j+1; M_1, M_2, \dots, M_m, P_1, \dots, P_{i-1}, P_i + p_{j+1}, P_{i+1}, \dots, P_m, Z + s_i + P_i + p_{j+1}, \max\{T, s_i + P_i + p_{j+1} - C_{j+1}(\pi^*)\}, R)$  is true for the combination  $(s_1, s_2, \dots, s_m)$ . For the picked representative state  $(j; M'_1, M'_2, \dots, M'_m, P'_1, P'_2, \dots, P'_m, Z', T', R')$ , one sees that the state  $(j+1; M'_1, M'_2, \dots, M'_m, P'_1, \dots, P'_{i-1}, P'_i + p_{j+1}, P'_{i+1}, \dots, P'_m, Z' + s'_i + P'_i + p_{j+1}, \max\{T', s'_i + P'_i + p_{j+1} - C_{j+1}(\pi^*)\}, R')$  is evaluated by the algorithm  $A(\epsilon)$  to be true for the combination  $(s'_1, s'_2, \dots, s'_m) \in \mathcal{S}$ .<sup>6</sup> For the true grid to which this state belongs, the picked representative state is denoted as  $(j+1; M''_1, M''_2, \dots, M''_m, P''_1, P''_2, \dots, P''_m, Z'', T'', R'')$  and satisfies

$$\begin{cases} s'_\ell < \delta^j s_\ell, \\ M''_\ell \leq \delta M'_\ell, \\ P''_i \leq \delta(P'_i + p_{j+1}), \\ P''_\ell \leq \delta P'_\ell, \text{ if } \ell \neq i, \\ Z'' \leq \delta(Z' + s'_i + P'_i + p_{j+1}), \\ T'' \leq \delta \max\{T', s'_i + P'_i + p_{j+1} - C_{j+1}(\pi^*)\}, \\ R'' \leq R', \end{cases}$$

Using

$$\begin{aligned} P'_i + p_{j+1} &\leq \delta^j P_i + p_{j+1} < \delta^j (P_i + p_{j+1}), \\ Z' + s'_i + P'_i + p_{j+1} &\leq \delta^j Z + \delta^j s_i + \delta^j P_i + p_{j+1} < \delta^j (Z + s_i + P_i + p_{j+1}), \\ \max\{T', s'_i + P'_i + p_{j+1} - C_{j+1}(\pi^*)\} \\ &\leq \max\{\delta^j T + (\delta^j - 1)Z, \delta^j s_i + \delta^j P_i + p_{j+1} - C_{j+1}(\pi^*)\} \\ &\leq \begin{cases} \max\{\delta^j T + (\delta^j - 1)Z, \delta^j (s_i + P_i) - (s_i + P_i)\}, & \text{if } s_i + P_i + p_{j+1} \leq C_{j+1}(\pi^*), \\ \max\{\delta^j T + (\delta^j - 1)Z, \delta^j (s_i + P_i + p_{j+1} - C_{j+1}(\pi^*)) + (\delta^j - 1)(s_i + P_i + p_{j+1})\}, & \text{otherwise,} \end{cases} \\ &\leq \begin{cases} \delta^j T + (\delta^j - 1)(Z + s_i + P_i), & \text{if } s_i + P_i + p_{j+1} \leq C_{j+1}(\pi^*), \\ \delta^j \max\{T, s_i + P_i + p_{j+1} - C_{j+1}(\pi^*)\} + (\delta^j - 1)(Z + s_i + P_i + p_{j+1}), & \text{otherwise,} \end{cases} \\ &\leq \delta^j \max\{T, s_i + P_i + p_{j+1} - C_{j+1}(\pi^*)\} + (\delta^j - 1)(Z + s_i + P_i + p_{j+1}), \end{aligned}$$

we have

$$\begin{cases} s'_\ell < \delta^j s_\ell, \\ M''_\ell \leq \delta M'_\ell, \\ P''_i \leq \delta(P'_i + p_{j+1}), \\ P''_\ell \leq \delta P'_\ell, \text{ if } \ell \neq i, \\ Z'' \leq \delta(Z' + s'_i + P'_i + p_{j+1}), \\ T'' \leq \delta \max\{T', s'_i + P'_i + p_{j+1} - C_{j+1}(\pi^*)\}, \\ R'' \leq R', \end{cases} + \begin{cases} M'_\ell \leq \delta^j M_\ell, \\ P'_\ell \leq \delta^j P_\ell, \\ Z' \leq \delta^j Z, \\ T' \leq \delta^j T + (\delta^j - 1)Z, \\ R' \leq R, \end{cases}$$

<sup>6</sup> In Case 3, no new combination of  $\mathcal{S}$  is involved.

$$\Rightarrow \begin{cases} s'_\ell < \delta^j s_\ell, \\ M''_\ell \leq \delta^{j+1} M_\ell, \\ P''_i \leq \delta^{j+1} (P_i + p_{j+1}), \\ P''_\ell \leq \delta^{j+1} P_\ell, \text{ if } \ell \neq i, \\ Z'' \leq \delta^{j+1} (Z + s_i + P_i + p_{j+1}), \\ T'' \leq \delta^{j+1} \max\{T, s_i + P_i + p_{j+1} - C_{j+1}(\pi^*)\} + (\delta^{j+1} - 1) \\ \quad (Z + s_i + P_i + p_{j+1}), \\ R'' \leq R; \end{cases}$$

that is, the statement holds for the state  $(j+1; M_1, M_2, \dots, M_m, P_1, \dots, P_{i-1}, P_i + p_{j+1}, P_{i+1}, \dots, P_m, Z + s_i + P_i + p_{j+1}, \max\{T, s_i + P_i + p_{j+1} - C_{j+1}(\pi^*)\}, R)$ . We therefore finish the proof of the inductive statement.

Next, we use the inequality  $\delta^n = (1 + \epsilon/2n)^n \leq 1 + \epsilon$  for any  $0 < \epsilon < 1$  to bound the performance ratio of the algorithm  $A(\epsilon)$ . Assume the state  $(n, M_1, M_2, \dots, M_m, P_1, P_2, \dots, P_m, Z, T, R)$  is true for the combination  $(s_1, s_2, \dots, s_m)$  of start processing times by the exact dynamic programming, with its objective value  $Z + R + \mu T$  and total rejection cost  $R \leq h$ . The above inductive statement says that the algorithm  $A(\epsilon)$  picks a true state  $(n, M'_1, M'_2, \dots, M'_m, P'_1, P'_2, \dots, P'_m, Z', T', R')$  for some combination  $(s'_1, s'_2, \dots, s'_m) \in \mathcal{S}$  as a representative with its total rejection cost  $R' \leq R \leq h$  and its objective value

$$\begin{aligned} & Z' + R' + \mu T' \\ & \leq \delta^n Z + R + \mu(\delta^n T + (\delta^n - 1)Z) \\ & \leq \delta^n (Z + R + \mu T) + \mu(\delta^n - 1)Z \\ & \leq (1 + \epsilon + \mu\epsilon)(Z + R + \mu T). \end{aligned}$$

Therefore, by re-scaling  $\epsilon$  to be  $\frac{\epsilon}{1+\mu}$ , the algorithm  $A(\epsilon)$  is a  $(1 + \epsilon)$ -approximation.

(We remark that the maximum tardiness can be increased up to  $(\delta^n - 1)(T + Z) \approx \epsilon(T + Z)$ , and thus not guaranteed bounded.)

For the time complexity, there are  $O(\prod_{\ell=1}^{2m+3} v_\ell)$  grids associated with each combination of start processing times and  $j$ , and for every grid the algorithm saves at most one true state, which leads to up to  $2m + 1$  true states. Using  $\epsilon/2 \leq \log(1 + \epsilon) \leq \epsilon$  for  $0 < \epsilon \leq 1$ , the overall running time of the algorithm  $A(\epsilon)$  is in  $O((2m + 1)nv_0^m \prod_{\ell=1}^{2m+3} v_\ell) \subseteq O((2m + 1)2^{6m+6}n^{3m+4}(\log M)^{3m+3}/\epsilon^{3m+3})$ , which is polynomial, where  $M = \max\{n, r_D, p_{\max}, h\}$ .  $\square$

## 5 Numerical experiments

From Theorem 1 we have a dynamic programming exact algorithm for computing an optimal reschedule for Problem (5), which has a space complexity of  $n^{m+1}(r_D + p_{\max})^m p_{\max}^{2m} k \min\{2n^2 p_{\max}, h\}$ . In 2018, Wang et al. have presented a similar algorithm for minimizing the total completion time and their algorithm has a lower space complexity. Yet, Wang et al. implemented their algorithm and showed that it works practically only on tiny instances. Our algorithm also has a too high space complexity for truthful implementation.

On the other hand, during the recurrence development we have observed that for each job there are only  $2m + 1$  possibilities: to be rejected, or to be assigned to one of the  $m$  earlier schedules and the  $m$  later schedules. This actually gives rise to an enumeration algorithm of  $(2m + 1)^n$  space complexity. One sees that despite  $(2m + 1)^n$  being exponential while  $n^{m+1}(r_{\mathcal{D}} + p_{\max})^m p_{\max}^{2m} k \min\{2n^2 p_{\max}, h\}$  being pseudo-polynomial, for reasonable ranges of these parameters the enumeration algorithm could be more practical.

These observations inspire us to implement the enumeration algorithm, but taking advantages of the dynamic programming recurrences. At the high level, we continue to use the states defined inside the dynamic programming algorithm; but we only compute those true states sequentially for the job subsets  $[j]$ , from  $j = 0, 1, \dots$  onward to  $n$ . Throughout the computation, we similarly ignore the individual job information about whether it is accepted or not, and if accepted, which earlier or later schedule it is assigned to; instead, we track for each partial schedule the  $2m$  total processing times, the total completion time, the maximum tardiness and the total rejection cost. We adopt the depth-first-search (DFS) to walk through the space of partial schedules, holding in hand the best reschedule obtained so far together with its objective value.

We denote our alternative enumeration algorithm as DFS- DP, and implemented it in C to validate the practical performance. One sees that the space complexity for DFS- DP is only  $O(n)$ .

## 5.1 Technical implementation details

Through the tracing of an illustrating instance in Sect. 4.2, we have learned the following technical details for implementing the DFS- DP algorithm:

1. Given  $\mathcal{D}$ ,  $r_{\mathcal{D}}$ ,  $h$  and  $k$ , annotations for individual jobs can be made on whether it should be rejected or should be accepted, and if accepted then to which ones of the earlier and the later schedules it can be assigned (see for example Table 1). Such annotations are effective in reducing the DFS space. We note that joint annotations may be made for a subset of jobs for further speedup, but developing such rules is probably too much work.
2. The start processing time combination in which  $s_i = r_{\mathcal{D}}$  for all  $i$  is likely the best one to go with, and the achieved minimum objective value can be used for earlier pruning partial schedules associated with the other combinations. For the illustrating instance, the minimum objective value for the combination (50, 50) is 278, which actually prunes away states 104 and 173, among others.
3. Note that each  $s_i > r_{\mathcal{D}}$  requires a subset of jobs (which are assigned to the earlier schedule on machine  $i$ ) of total processing time equal to  $s_i$ . We thus can run one time the dynamic programming algorithm for Subset-Sum to determine all the possible  $s_i$  values. For the same reason, we can track the total space in the earlier schedules that has to be filled, which is  $\sum_{i: s_i > r_{\mathcal{D}}} (s_i - M_i)$ , and the total processing time of the remaining jobs that can be assigned to earlier schedules; if the former is strictly larger than the latter, then the state does not lead to a feasible reschedule.

## 5.2 Instance simulation

We adapt the data simulation scheme used in the numerical experiments by Wang et al. (2018), as follows. Recall that all numbers in an instance are non-negative integers.

- The number of machines is  $m \in \{2, 3, 4, 6, 8\}$ ;
- the number of jobs is  $n \in \{4, 6, 8, \dots, 20\}$  (increment by 2) and such that  $n \geq m$ ;
- the job processing times are  $p_j \sim [1, 30]$  (random uniform distribution);
- the number of delayed jobs is set to one of  $\{0.1n, 0.2n, 0.5n\}$ ;
- the release date is  $r_D \in \{\frac{1}{3} \sum_{j=1}^n p_j, \frac{1}{2} \sum_{j=1}^n p_j, \frac{2}{3} \sum_{j=1}^n p_j\}$ ;
- the maximum tardiness upper bound is  $k \in \{\frac{1}{2} \sum_{j=1}^n p_j, \frac{5}{8} \sum_{j=1}^n p_j, \frac{3}{4} \sum_{j=1}^n p_j\}$ ;
- the tardiness scaling factor is  $\mu \in \{1, 10, 100\}$ ;
- the job rejection costs are  $e_j = k_j p_j$ , where  $k_j \sim [1, 10]$  (random uniform distribution);
- total rejection cost upper bound is  $h \in \{\frac{1}{3} \sum_{j=1}^n e_j, \frac{1}{2} \sum_{j=1}^n e_j, \frac{2}{3} \sum_{j=1}^n e_j\}$ .

For each pair  $(m, n)$ , the simulation process is repeated 20 times to generate 20 instances. The experiments were run on a number of identical Linux boxes each with four 3.0GHz processors and 32GB memory. We set the running time limit for each instance at 2 h, and force termination when time is up.

## 5.3 Results

Recall that for each pair  $(m, n)$ , we have 20 independent simulated instances. For each instance, we calculated the estimated density of the true states in the entire state space in the dynamic programming exact algorithm using the ratio  $(2m + 1)^n / (n^{m+1} (r_D + p_{\max})^m p_{\max}^m k \min\{2n^2 p_{\max}, h\})$ , and the highest density among the 20 instances is associated with the pair  $(m, n)$ , in Table 3. The performance of DFS- DP on the pair  $(m, n)$  is measured as the average elapsed real (running) time over the 20 instances (collected in Table 3), excluding those on which DFS- DP didn't finish in 2 h.

In Table 3, the only instance with 12 jobs on which DFS- DP didn't finish in 2 h is for 8 machines, and the elapsed real time for DFS- DP is slightly longer than 8 h. For all the 100 instances with 10 jobs, we ran DFS- DP with  $m = 2, 3, 4, 6, 8$  machines and plot the elapsed real times in Fig. 1, where the instances are sorted in the increasing real time order when  $m = 8$  machines are deployed. In the plot, two points for 8 machines escape and the two real times are 323.51 and 600.31 s, respectively.

From the plot, one can see that the “difficulty” level of an instance is seemingly inherent, mostly attributed to the simulated values for the instance parameters; and, understandably, the difficulty level increases along with the number of machines.

## 5.4 Discussion

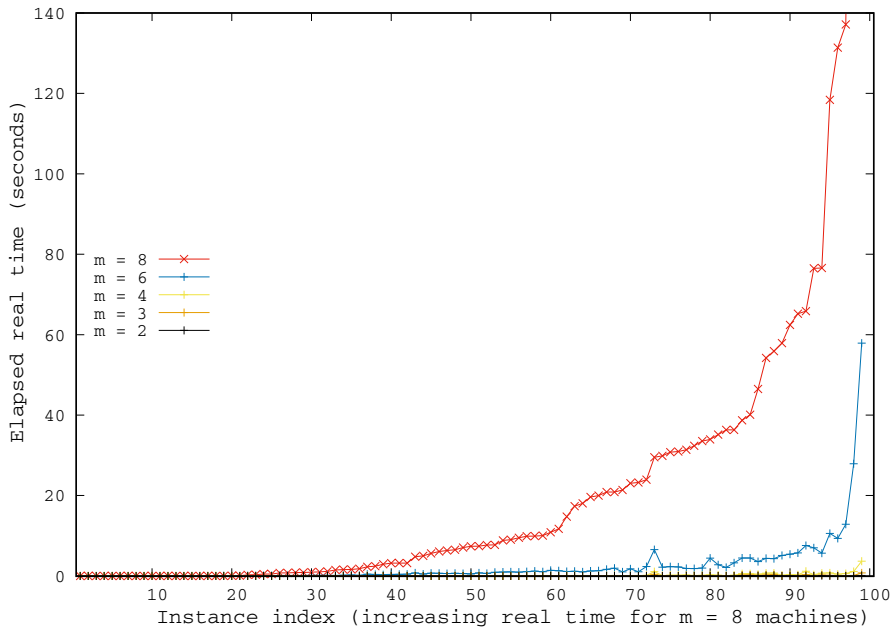
Wang et al. (2018) showed very poor performance for their dynamic programming exact algorithm, likely due to the faithful implementation as we encountered the same issue. The DFS- DP exact algorithm seems to be able to solve instances of reasonable sizes; it takes advantages of both the dynamic programming recurrences and the fact

**Table 3** Highest estimated true state density and the performance of DFS-DP in the average elapsed real time over 20 randomly generated instances, for each pair of  $(m, n)$

$n$	$m = 2$	3		4		6		8		
	Density	Time	Density	Time	Density	Time	Density	Time	Density	Time
4	$10^{-12}$	00:00.00	$10^{-16}$	00:00.00	$10^{-20}$	00:00.00	—	—	—	—
6	$10^{-11}$	00:00.00	$10^{-18}$	00:00.00	$10^{-23}$	00:00.00	$10^{-30}$	00:00.00	—	—
8	$10^{-13}$	00:00.00	$10^{-18}$	00:00.00	$10^{-22}$	00:00.01	$10^{-30}$	00:00.11	$10^{-39}$	00:00.79
10	$10^{-14}$	00:00.01	$10^{-15}$	00:00.03	$10^{-20}$	00:00.09	$10^{-33}$	00:06.03	$10^{-43}$	00:19.52
12	$10^{-12}$	00:00.15	$10^{-18}$	00:00.46	$10^{-21}$	00:02.20	$10^{-30}$	01:06.04	$10^{-41}$	02:18.95 <sup>1</sup>
14	$10^{-13}$	00:00.52	$10^{-16}$	00:01.04	$10^{-21}$	01:20.01	$10^{-31}$	06:42.61 <sup>1</sup>	$10^{-40}$	17:28.18 <sup>10</sup>
16	$10^{-12}$	00:06.84	$10^{-16}$	01:10.79	$10^{-20}$	03:28.03	$10^{-29}$	29:15.91 <sup>11</sup>	$10^{-39}$	05:42.87 <sup>14</sup>
18	$10^{-10}$	00:22.22	$10^{-14}$	14:54.56	$10^{-17}$	40:05.18 <sup>4</sup>	$10^{-28}$	38:26.33 <sup>15</sup>	$10^{-39}$	19:03.02 <sup>11</sup>
20	$10^{-10}$	04:45.07	$10^{-13}$	05:39.40 <sup>4</sup>	$10^{-17}$	27:10.45 <sup>11</sup>	$10^{-27}$	52:36.80 <sup>15</sup>	$10^{-37}$	02:34.28 <sup>15</sup>

It was observed that the peak memory usage for each instance is less than 2GB. The standard time format hh:mm:ss.ff is hh hours, mm minutes, and ss.ff seconds. An entry with a superscript “x” indicates x out of the 20 instances didn’t terminate within the 2-h time limit, and the average is over the other  $20 - x$  instances





**Fig. 1** The elapsed real times for DFS-DP on the 100 instances with 10 jobs, when  $m = 2, 3, 4, 6, 8$  machines are deployed respectively. The instances are sorted in the increasing real time order when  $m = 8$  machines are deployed. Two points for  $m = 8$  machines escape from the plot and the two real times are 323.51 and 600.31 s, respectively

that true states are very sparse in the entire state space in the dynamic programming exact algorithm. Our numerical experiments validated that such a choice combined with DFS gives rise to very good practical performance; and it can be another working idea for solving similar problems which admit pseudo-polynomial time dynamic programming exact algorithms.

## 6 Conclusions and future work

In this paper, we examined the rejection-allowed multiprocessor rescheduling problem to respond to job delays, with the objective to minimize the sum of the total completion time of the accepted jobs, the total rejection cost of the rejected jobs, and the penalty on the maximum tardiness of the accepted jobs, subject to an upper bound on the total rejection cost and an upper bound on the maximum tardiness. We studied the problem from the approximation algorithm perspective, aiming to address an open problem left in Luo et al. (2020). In the route, we first observed several structural properties associated with optimal reschedules, and then used them to design the recurrences between feasible partial reschedules; we developed a pseudo-polynomial time dynamic programming exact algorithm based on these recurrences. When the upper bound on the maximum tardiness is lifted, we used the standard sparsing technique to transfer the exact algorithm into an FPTAS. Given the NP-hardness of the problem, we have there-

fore achieved the best possible approximability result when the maximum tardiness can be unbounded, and answered affirmatively partially the open problem left in Luo et al. (2020) when the number of parallel identical machines is fixed.

We supplemented the theoretical studies with numerical experiments to demonstrate the performance of the proposed exact algorithm. We observed that the dynamic programming exact algorithm has a space complexity too high for truthful implementation, and suggested an alternative to integrate the recurrences and the enumeration, followed by a DFS walk in the space of the feasible partial reschedules. The numerical experiments show its promising performance.

For the general case where the maximum tardiness is upper bounded by a given constant, it seems challenging to develop the pseudo-polynomial time dynamic programming exact algorithm into an FPTAS or a PTAS. Our observation or at least our design limitation is that, in a sparsing scheme, it is difficult to bound both the maximum tardiness and the total rejection cost. Nevertheless, an FPTAS or a PTAS is perhaps still possible, but not necessarily through the dynamic programming exact algorithm. On the other hand, if a hardness proof can be established to show the non-existence of a PTAS, then it would be interesting to examine the existence of an FPTAS or a PTAS when the maximum tardiness is upper bounded but the upper bound on the total rejection cost is lifted, that is, the dual case to where we have succeeded. In fact, this is the case for the single machine rescheduling problem examined in Luo et al. (2020), for which an FPTAS is achieved.

Another direction for research would be to relax the strict upper bound on the maximum tardiness, such as  $f(\epsilon) \cdot k$  for some polynomial time computable function  $f(\epsilon)$ , and to ask whether an FPTAS or a PTAS exists. Note that we have not yet answered this question, since the maximum tardiness in our solution can increase up to  $\epsilon k + \epsilon Z^*$ , where  $Z^*$  is the total completion time in an optimal reschedule but has no seemingly direct relationship with  $k$ . Lastly, one can replace one or both the hard constraints on the maximum tardiness and the total rejection cost by the corresponding objectives, leading to bi-/tri-criteria optimization problems, and to study their approximabilities. For example, whether or not there exists an FPTAS or a PTAS returning a  $(1 + \epsilon)$ -approximate reschedule in which the maximum tardiness is no greater than  $f(\epsilon) \cdot k$  and the total rejection cost is no greater than  $g(\epsilon) \cdot h$ , for some functions  $f(\cdot)$  and  $g(\cdot)$ ?

**Acknowledgements** The authors are grateful to the handling editor and the reviewers for their helpful comments and suggestions. WL is supported by K. C. Wong Magna Fund in Ningbo University, the Humanities and Social Sciences Planning Foundation of the Ministry of Education (Grant No. 18YJA630077), Zhejiang Provincial Natural Science Foundation (Grant No. LY19A010005), the Ningbo Natural Science Foundation (Grant No. 2018A610198), and the National Natural Science Foundation of China (Grant No. 11971252). RC, AC, GL and AZ are supported by the NSERC Canada. BS is supported partially by the Humanities and Social Science Foundation of Ministry of Education of China (Grant No. 18YJAZH080) and Science and Technology Department of Shaanxi Province (Grant No. 2020JQ-654). AZ is supported by the National Natural Science Foundation of China (Grant Nos. 11971139 and 11771114) and the China Scholarship Council (Grant No. 201908330090).

**Data availability** The C programs implementing the DFS-DP algorithm and for random instance generation are available upon request, within the first three years.

Declarations

**Conflict of interests** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A: An illustration of the dynamic programming exact algorithm

Below is an illustration of executing the dynamic programming exact algorithm on the 10-job instance described in Table 1, for the combination  $(s_1, s_2) = (r_{\mathcal{D}} + 13, r_{\mathcal{D}}) = (63, 50)$ . The following Table 4 lists all the transitions from a true state with  $j$  to a true

**Table 4** The execution of the dynamic programming exact algorithm on the 10-job instance for the combination  $(s_1, s_2) = (63, 50)$ , listing all the transitions from  $j$  to  $j + 1$ , for  $j = 0, 1, 2, 3, 6, 7, 8, 9, 10$ , while using the job annotations from Table 1. In particular,  $j$  jumps from 3 to 6 since all the three jobs 4, 5, 6 are rejected

No.	True state	Source state	Comments/objective value
1	(0; 0, 0, 0, 0, 0, 0)		
2	(1; 0, 0, 0, 0, 0, 12)	1, Case 1	Job 1 can be accepted into earlier's
3	(1; 2, 0, 0, 0, 2, 0, 0)	1, Case 1	
4	(1; 0, 2, 0, 0, 2, 0, 0)	1, Case 1	
5	(2; 5, 0, 0, 0, 5, 0, 12)	2, Case 1	Job 2 is accepted into earlier's
6	(2; 0, 5, 0, 0, 5, 0, 12)	2, Case 1	
7	(2; 7, 0, 0, 0, 9, 2, 0)	3, Case 1	
8	(2; 2, 5, 0, 0, 7, 0, 0)	3, Case 1	
9	(2; 5, 2, 0, 0, 7, 0, 0)	4, Case 1	
10	(2; 0, 7, 0, 0, 9, 2, 0)	4, Case 1	
11	(3; 12, 0, 0, 0, 17, 3, 12)	5, Case 1	Job 3 is accepted into earlier's
12	(3; 5, 7, 0, 0, 12, 0, 12)	5, Case 1	
13	(3; 7, 5, 0, 0, 12, 0, 12)	6, Case 1	
14	(3; 0, 12, 0, 0, 17, 3, 12)	6, Case 1	
15	(3; 14, 0, 0, 0, 23, 5, 0)	7, Case 1	
16	(3; 7, 7, 0, 0, 16, 2, 0)	7, Case 1	
17	(3; 9, 5, 0, 0, 16, 0, 0)	8, Case 1	
18	(3; 2, 12, 0, 0, 19, 3, 0)	8, Case 1	
19	(3; 12, 2, 0, 0, 19, 3, 0)	9, Case 1	
20	(3; 5, 9, 0, 0, 16, 0, 0)	9, Case 1	
21	(3; 7, 7, 0, 0, 16, 2, 0)	10, Case 1	superior than State 16; 16 eliminated
22	(3; 0, 14, 0, 0, 23, 5, 0)	10, Case 1	
23	(6; 12, 0, 0, 0, 17, 3, 80)	11, Case 2	Jobs 4, 5, 6 are rejected
24	(6; 5, 7, 0, 0, 12, 0, 80)	12, Case 2	
25	(6; 7, 5, 0, 0, 12, 0, 80)	13, Case 2	

**Table 4** continued

No.	True state	Source state	Comments/objective value
26	(6; 0, 12, 0, 0, 17, 3, 80)	14, Case 2	
27	(6; 14, 0, 0, 0, 23, 5, 68)	15, Case 2	
28	(6; 7, 7, 0, 0, 16, 2, 68)	16, Case 2	
29	(6; 9, 5, 0, 0, 16, 0, 68)	17, Case 2	
30	(6; 2, 12, 0, 0, 19, 3, 68)	18, Case 2	
31	(6; 12, 2, 0, 0, 19, 3, 68)	19, Case 2	
32	(6; 5, 9, 0, 0, 16, 0, 68)	20, Case 2	
33	(6; 0, 14, 0, 0, 23, 5, 68)	22, Case 2	
34	(7; 40, 0, 0, 0, 57, 3, 80)	23, Case 1	Job 7 is accepted into earlier's
35	(7; 12, 28, 0, 0, 45, 3, 80)	23, Case 1	
36	(7; 33, 7, 0, 0, 45, 0, 80)	24, Case 1	
37	(7; 5, 35, 0, 0, 47, 0, 80)	24, Case 1	
38	(7; 35, 5, 0, 0, 47, 0, 80)	25, Case 1	
39	(7; 7, 33, 0, 0, 45, 0, 80)	25, Case 1	
40	(7; 28, 12, 0, 0, 45, 3, 80)	26, Case 1	
41	(7; 0, 40, 0, 0, 57, 3, 80)	26, Case 1	
42	(7; 42, 0, 0, 0, 65, 5, 68)	27, Case 1	
43	(7; 14, 28, 0, 0, 51, 5, 68)	27, Case 1	
44	(7; 35, 7, 0, 0, 51, 2, 68)	28, Case 1	
45	(7; 7, 35, 0, 0, 51, 2, 68)	28, Case 1	
46	(7; 37, 5, 0, 0, 53, 0, 68)	29, Case 1	
47	(7; 9, 33, 0, 0, 49, 0, 68)	29, Case 1	
48	(7; 30, 12, 0, 0, 49, 3, 68)	30, Case 1	
49	(7; 2, 40, 0, 0, 59, 3, 68)	30, Case 1	
50	(7; 40, 2, 0, 0, 59, 3, 68)	31, Case 1	
51	(7; 12, 30, 0, 0, 49, 3, 68)	31, Case 1	
52	(7; 33, 9, 0, 0, 49, 0, 68)	32, Case 1	
53	(7; 5, 37, 0, 0, 53, 0, 68)	32, Case 1	
54	(7; 28, 14, 0, 0, 51, 5, 68)	33, Case 1	
55	(7; 0, 42, 0, 0, 65, 5, 68)	33, Case 1	
56	(8; 40, 0, 0, 30, 137, 13, 80)	34, Case 2	Job 8 can be accepted into later 2
57	(8; 12, 28, 0, 30, 125, 13, 80)	35, Case 2	
58	(8; 33, 7, 0, 30, 125, 13, 80)	36, Case 2	
59	(8; 5, 35, 0, 30, 127, 13, 80)	37, Case 2	
60	(8; 35, 5, 0, 30, 127, 13, 80)	38, Case 2	

**Table 4** continued

No.	True state	Source state	Comments/objective value
61	(8; 7, 33, 0, 30, 125, 13, 80)	39, Case 2	
62	(8; 28, 12, 0, 30, 125, 13, 80)	40, Case 2	
63	(8; 0, 40, 0, 30, 137, 13, 80)	41, Case 2	
64	(8; 42, 0, 0, 0, 65, 5, 93)	42, Case 2	
65	(8; 42, 0, 0, 30, 145, 13, 68)	42, Case 2	
66	(8; 14, 28, 0, 0, 51, 5, 93)	43, Case 2	
67	(8; 14, 28, 0, 30, 131, 13, 68)	43, Case 2	
68	(8; 35, 7, 0, 0, 51, 2, 93)	44, Case 2	
69	(8; 35, 7, 0, 30, 131, 13, 68)	44, Case 2	
70	(8; 7, 35, 0, 0, 51, 2, 93)	45, Case 2	
71	(8; 7, 35, 0, 30, 131, 13, 68)	45, Case 2	
72	(8; 37, 5, 0, 0, 53, 0, 93)	46, Case 2	
73	(8; 37, 5, 0, 30, 133, 13, 68)	46, Case 2	
74	(8; 9, 33, 0, 0, 49, 0, 93)	47, Case 2	
75	(8; 9, 33, 0, 30, 129, 13, 68)	47, Case 2	
76	(8; 30, 12, 0, 0, 49, 3, 93)	48, Case 2	
77	(8; 30, 12, 0, 30, 129, 13, 68)	48, Case 2	
78	(8; 2, 40, 0, 0, 59, 3, 93)	49, Case 2	
79	(8; 2, 40, 0, 30, 139, 13, 68)	49, Case 2	
80	(8; 40, 2, 0, 0, 59, 3, 93)	50, Case 2	
81	(8; 40, 2, 0, 30, 139, 13, 68)	50, Case 2	
82	(8; 12, 30, 0, 0, 49, 3, 93)	51, Case 2	
83	(8; 12, 30, 0, 30, 129, 13, 68)	51, Case 2	
84	(8; 33, 9, 0, 0, 49, 0, 93)	52, Case 2	
85	(8; 33, 9, 0, 30, 129, 13, 68)	52, Case 2	
86	(8; 5, 37, 0, 0, 53, 0, 93)	53, Case 2	
87	(8; 5, 37, 0, 30, 133, 13, 68)	53, Case 2	
88	(8; 28, 14, 0, 0, 51, 5, 93)	54, Case 2	
89	(8; 28, 14, 0, 30, 131, 13, 68)	54, Case 2	
90	(8; 0, 42, 0, 0, 65, 5, 93)	55, Case 2	
91	(8; 0, 42, 0, 30, 145, 13, 68)	55, Case 2	
92	(9; 40, 35, 0, 30, 172, 13, 80)	56, Case 1	Job 9 can be accepted
93	(9; 40, 0, 35, 30, 235, 13, 80)	56, Case 1	
94	(9; 47, 28, 0, 30, 172, 13, 80)	57, Case 1	
95	(9; 12, 28, 35, 30, 223, 13, 80)	57, Case 1	
96	(9; 33, 42, 0, 30, 167, 13, 80)	58, Case 1	
97	(9; 33, 7, 35, 30, 223, 13, 80)	58, Case 1	
98	(9; 40, 35, 0, 30, 167, 13, 80)	59, Case 1	superior than State 92; 92 eliminated
99	(9; 5, 35, 35, 30, 225, 13, 80)	59, Case 1	

**Table 4** continued

No.	True state	Source state	Comments/objective value
100	(9; 35, 40, 0, 30, 167, 13, 80)	60, Case 1	
101	(9; 35, 5, 35, 30, 225, 13, 80)	60, Case 1	
102	(9; 42, 33, 0, 30, 167, 13, 80)	61, Case 1	
103	(9; 7, 33, 35, 30, 223, 13, 80)	61, Case 1	
104	(9; 63, 12, 0, 30, 188, 13, 80)	62, Case 1	
105	(9; 28, 47, 0, 30, 172, 13, 80)	62, Case 1	
106	(9; 28, 12, 35, 30, 270, 13, 80)	62, Case 1	
107	(9; 35, 40, 0, 30, 172, 13, 80)	63, Case 1	inferior than State 100; eliminated
108	(9; 0, 40, 35, 30, 235, 13, 80)	63, Case 1	
109	(9; 42, 35, 0, 0, 100, 5, 93)	64, Case 1	
110	(9; 42, 0, 35, 0, 163, 5, 93)	64, Case 1	
111	(9; 42, 0, 0, 35, 145, 5, 93)	64, Case 1	
112	(9; 42, 35, 0, 30, 180, 13, 68)	65, Case 1	
113	(9; 42, 0, 35, 30, 243, 13, 68)	65, Case 1	
114	(9; 49, 28, 0, 0, 100, 5, 93)	66, Case 1	
115	(9; 14, 28, 35, 0, 149, 5, 93)	66, Case 1	
116	(9; 14, 28, 0, 35, 131, 5, 93)	66, Case 1	
117	(9; 49, 28, 0, 30, 180, 13, 68)	67, Case 1	
118	(9; 14, 28, 35, 30, 229, 13, 68)	67, Case 1	
119	(9; 35, 42, 0, 0, 93, 2, 93)	68, Case 1	
120	(9; 35, 7, 35, 0, 149, 2, 93)	68, Case 1	
121	(9; 35, 7, 0, 35, 131, 2, 93)	68, Case 1	
122	(9; 35, 42, 0, 30, 173, 13, 68)	69, Case 1	
123	(9; 35, 7, 35, 30, 229, 13, 68)	69, Case 1	
124	(9; 42, 35, 0, 0, 93, 2, 93)	70, Case 1	
125	(9; 7, 35, 35, 0, 149, 2, 93)	70, Case 1	
126	(9; 7, 35, 0, 35, 131, 2, 93)	70, Case 1	
127	(9; 42, 35, 0, 30, 173, 13, 68)	71, Case 1	
128	(9; 7, 35, 35, 30, 229, 13, 68)	71, Case 1	
129	(9; 37, 40, 0, 0, 93, 0, 93)	72, Case 1	
130	(9; 37, 5, 35, 0, 151, 0, 93)	72, Case 1	
131	(9; 37, 5, 0, 35, 133, 0, 93)	72, Case 1	
132	(9; 37, 40, 0, 30, 173, 13, 68)	73, Case 1	
133	(9; 37, 5, 35, 30, 231, 13, 68)	73, Case 1	
134	(9; 44, 33, 0, 0, 93, 0, 93)	74, Case 1	

**Table 4** continued

No.	True state	Source state	Comments/objective value
135	(9; 9, 33, 35, 0, 147, 0, 93)	74, Case 1	
136	(9; 9, 33, 0, 35, 129, 0, 93)	74, Case 1	
137	(9; 44, 33, 0, 30, 173, 13, 68)	75, Case 1	
138	(9; 9, 33, 35, 30, 227, 13, 68)	75, Case 1	
139	(9; 30, 47, 0, 0, 96, 3, 93)	76, Case 1	
140	(9; 30, 12, 35, 0, 147, 3, 93)	76, Case 1	
141	(9; 30, 12, 0, 35, 129, 3, 93)	76, Case 1	
142	(9; 30, 47, 0, 30, 176, 13, 68)	77, Case 1	
143	(9; 30, 12, 35, 30, 227, 13, 68)	77, Case 1	
144	(9; 37, 40, 0, 0, 96, 3, 93)	78, Case 1	
145	(9; 2, 40, 35, 0, 157, 3, 93)	78, Case 1	
146	(9; 2, 40, 0, 35, 139, 3, 93)	78, Case 1	
147	(9; 37, 40, 0, 30, 176, 13, 68)	79, Case 1	
148	(9; 2, 40, 35, 30, 237, 13, 68)	79, Case 1	
149	(9; 40, 37, 0, 0, 96, 3, 93)	80, Case 1	
150	(9; 40, 2, 35, 0, 157, 3, 93)	80, Case 1	
151	(9; 40, 2, 0, 35, 139, 3, 93)	80, Case 1	
152	(9; 40, 37, 0, 30, 176, 13, 68)	81, Case 1	
153	(9; 40, 2, 35, 30, 237, 13, 68)	81, Case 1	
154	(9; 47, 30, 0, 0, 96, 3, 93)	82, Case 1	
155	(9; 12, 30, 35, 0, 147, 3, 93)	82, Case 1	
156	(9; 12, 30, 0, 35, 129, 3, 93)	82, Case 1	
157	(9; 47, 30, 0, 30, 176, 13, 68)	83, Case 1	
158	(9; 12, 30, 35, 30, 227, 13, 68)	83, Case 1	
159	(9; 33, 44, 0, 0, 93, 0, 93)	84, Case 1	
160	(9; 33, 9, 35, 0, 147, 0, 93)	84, Case 1	
161	(9; 33, 9, 0, 35, 129, 0, 93)	84, Case 1	
162	(9; 33, 44, 0, 30, 173, 13, 68)	85, Case 1	
163	(9; 33, 9, 35, 30, 227, 13, 68)	85, Case 1	
164	(9; 40, 37, 0, 0, 93, 0, 93)	86, Case 1	
165	(9; 5, 37, 35, 0, 151, 0, 93)	86, Case 1	
166	(9; 5, 37, 0, 35, 133, 0, 93)	86, Case 1	
167	(9; 40, 37, 0, 30, 173, 13, 68)	87, Case 1	

state with  $j + 1$ , for  $j = 0, 1, 2, 3, 6, 7, 8, 9, 10$ , in each of which we give every true state a number, and the third column records from which true state and in which case it is first time derived true, except the initial true state. The last column contains additional comments, if any, for possibly better implementations of the exact algorithms (see Sect. 5), and the objective values for those true states ( $n; M_1, M_2, P_1, P_2, Z, T, R$ ) with  $s_i = \max\{M_i, r_D\}$  for each machine  $i$ .

**Table 4** continued

No.	True state	Source state	Comments/objective value
168	(9; 5, 37, 35, 30, 231, 13, 68)	87, Case 1	
169	(9; 63, 14, 0, 0, 114, 5, 93)	88, Case 1	
170	(9; 28, 49, 0, 0, 100, 5, 93)	88, Case 1	
171	(9; 28, 14, 35, 0, 149, 5, 93)	88, Case 1	
172	(9; 28, 14, 0, 35, 131, 5, 93)	88, Case 1	
173	(9; 63, 14, 0, 30, 194, 13, 68)	89, Case 1	
174	(9; 28, 49, 0, 30, 180, 13, 68)	89, Case 1	
175	(9; 28, 14, 35, 30, 229, 13, 68)	89, Case 1	
176	(9; 35, 42, 0, 0, 100, 5, 93)	90, Case 1	
177	(9; 0, 42, 35, 0, 163, 5, 93)	90, Case 1	
178	(9; 0, 42, 0, 35, 145, 5, 93)	90, Case 1	
179	(9; 35, 42, 0, 30, 180, 13, 68)	91, Case 1	
180	(9; 0, 42, 35, 30, 243, 13, 68)	91, Case 1	
181	(10; 63, 12, 42, 30, 293, 13, 80)	104, Case 2	Job 10 is accepted into later's/412
182	(10; 63, 12, 0, 72, 310, 13, 80)	104, Case 2	/429
183	(10; 63, 14, 42, 0, 219, 5, 93)	169, Case 2	/327
184	(10; 63, 14, 0, 42, 206, 5, 93)	169, Case 2	/314
185	(10; 63, 14, 42, 30, 299, 13, 68)	173, Case 2	/406
186	(10; 63, 14, 0, 72, 316, 13, 68)	173, Case 2	/423

There are six true states of the form (10; 63, ·, ·, ·, ·, ·, ·) corresponding to six feasible reschedules under the constraint  $(s_1, s_2) = (63, 50)$ , respectively, and the minimum objective value is 314. One can trace back the optimal reschedule in which the earlier schedule on machine 1 processes the jobs 7, 9, the later schedule on machine 1 is empty, the earlier schedule on machine 2 processes the jobs 1, 2, 3, and the later schedule on machine 2 processes the job 10.

## References

- Addis B, Carello G, Grosso A, Tànfani E (2016) Operating room scheduling and rescheduling: a rolling horizon approach. *Flex Serv Manuf J* 28:206–232
- Bartal Y, Leonardi S, Marchetti-Spaccamela A, Sgall J, Stougie L (2000) Multiprocessor scheduling with rejection. In: *Proceedings of ACM-IEEE SODA 2000*, pp 95–103
- Bean JC, Birge JR, Mittenthal J, Noon CE (1991) Matchup scheduling with multiple resources, release dates and disruptions. *Oper Res* 39:470–483
- Cheng Y, Sun S (2007) Scheduling linear deteriorating jobs with rejection on a single machine. *Eur J Oper Res* 194:18–27
- Clausen J, Larsen J, Larsen A, Hansen J (2001) Disruption management—operations research between planning and execution. *OR/MS Today* 28:40–43
- Dahal K, Al-Arfaj K, Paudyal K (2015) Modelling generator maintenance scheduling costs in deregulated power markets. *Eur J Oper Res* 240:551–561
- Garey MR, Johnson DS (1979) *Computers and intractability: a guide to the theory of np-completeness*. W. H. Freeman and Company, San Francisco



- Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan A, Rinnooy Kan A H G (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann Disc Math* 5:287–326
- Gupta JND, Ruiz-Torres AJ, Webster S (2003) Minimizing maximum tardiness and number of tardy jobs on parallel machines subject to minimum flow-time. *J Oper Res Soc* 54:1263–1274
- Hall NG, Liu Z, Potts CN (2007) Rescheduling for multiple new orders. *INFORMS J Comput* 19:633–645
- Hall NG, Potts CN (2004) Rescheduling for new orders. *Oper Res* 52:440–453
- Hall NG, Potts CN (2010) Rescheduling for job unavailability. *Oper Res* 58:746–755
- Li D, Lu X (2017) Two-agent parallel-machine scheduling with rejection. *Theoret Comput Sci* 703:66–75
- Liu Z, Lu L, Qi X (2018) Cost allocation in rescheduling with machine unavailable period. *Eur J Oper Res* 266:16–28
- Liu Z, Ro YK (2014) Rescheduling for machine disruption to minimize makespan and maximum lateness. *J Sched* 17:339–352
- Luo W, Jin M, Su B, Lin G (2020) An approximation scheme for rejection-allowed single-machine rescheduling. *Comput Indus Eng* 146 (Article 106574)
- Luo W, Luo T, Goebel R, Lin G (2018) Rescheduling due to machine disruption to minimize the total weighted completion time. *J Sched* 21:565–578
- Manavizadeh N, Goodarzi AH, Rabbani M, Jolai F (2013) Order acceptance/rejection policies in determining the sequence in mixed model assembly lines. *Appl Math Model* 37:2531–2551
- Ou J, Zhong X (2017) Bicriteria order acceptance and scheduling with consideration of fill rate. *Eur J Oper Res* 262:904–907
- Rinnooy Kan AHG (1976) Machine scheduling problems: classification, complexity, and computations. Springer
- Shabtay D (2014) The single machine serial batch scheduling problem with rejection to minimize total completion time and total rejection cost. *Eur J Oper Res* 233:64–74
- Unal AT, Uzsoy R, Kiran AS (1997) Rescheduling on a single machine with part-type dependent setup times and deadlines. *Ann Oper Res* 70:93–113
- Wang D, Yin Y, Cheng TCE (2018) Parallel-machine rescheduling with job unavailability and rejection. *Omega* 81:246–260
- Wang D-J, Liu F, Jin Y (2017) A multi-objective evolutionary algorithm guided by directed search for dynamic scheduling. *Comput Oper Res* 79:279–290
- Wu SD, Storer RH, Chang PC (1992) A rescheduling procedure for manufacturing systems under random disruptions. In: Fandel G, Gullledge T, Jones A (eds) *New directions for operations research in manufacturing*. Springer, Berlin, Heidelberg, pp 292–306
- Yin Y, Cheng TCE, Wang D, Wu CC (2016) Improved algorithms for single-machine serial-batch scheduling with rejection to minimize total completion time and total rejection cost. *IEEE Trans Syst Man Cybern Syst* 46:1578–1588
- Yin Y, Cheng TCE, Wang D-J (2016) Rescheduling on identical parallel machines with machine disruptions to minimize total completion time. *Eur J Oper Res* 252:737–749
- Yu G, Argüello M, Song G, McCowan SM, White A (2003) A new era for crew recovery at continental airlines. *INFORMS J Appl Anal* 33:5–22
- Zhang L, Lu L, Yuan J (2010) Single-machine scheduling under the job rejection constraint. *Theor Comput Sci* 411:1877–1882
- Zhang X, Lin W-C, Wu C-C (2021) Rescheduling problems with allowing for the unexpected new jobs arrival. *J Comb Optim*. <https://doi.org/10.1007/s10878-021-00803-4>
- Zhao Q, Lu L, Yuan J (2016) Rescheduling with new orders and general maximum allowable time disruptions. *4OR* 14:261–280
- Zweiben M, Davis E, Daun B, Deale MJ (1993) Scheduling and rescheduling with iterative repair. *IEEE Trans Syst Man Cybern* 23:1588–1596