A tardiness-augmented approximation scheme for rejection-allowed multiprocessors rescheduling

Wenchang Luo* Rylan Chin§ An Zhang† Alexander Cai§ Guohui Lin‡§

July 4, 2020

Abstract

In the multiprocessors rescheduling problem, a set of jobs has been planned in an original schedule to minimize their total completion time on a number of parallel identical machines, assuming but not with 100% certainty their availability at the beginning of the planned schedule. The need for rescheduling arises due to some jobs could not arrive in time; the decision-maker has to adjust the original schedule to account for the delayed jobs without causing excessive time disruption to the original schedule and to minimize their operational cost. While the decision-maker is allowed to reject any of the delayed or non-delayed jobs, the total rejection cost and the tardiness of each accepted job in the adjusted schedule are strictly upper bounded by given thresholds, respectively. We study a novel objective function that includes three cost components: the total completion time of the accepted jobs, the total rejection cost of the rejected jobs, and the penalty on the maximum tardiness for the accepted jobs. We study this novel rescheduling problem from approximation algorithm perspective, as it generalizes several classic NP-hard scheduling problems; we design a pseudo-polynomial time dynamic programming exact algorithm and, when the tardiness is unbounded, we develop the exact algorithm into a fully polynomial time approximation scheme. While both algorithms have high worst-case time and space complexity, we implemented them in C with careful memory allocation and the numerical experiments show very promising performance in terms of practical time and space complexity.

Keywords: Rescheduling; job delay; job rejection; dynamic programming; approximation scheme

Acknowledgements. W.L. 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). AZ is supported by the National Natural Science Foundation of China (Grant Nos. 11971139 and 11771114) and the China Scholarship Council (Grant No. 201908330090). RC, AZ, AC and GL are supported by the NSERC Canada.

*School of Mathematics and Statistics, Ningbo University. Ningbo, China. luowenchang@163.com

†Department of Mathematics, Hangzhou Dianzi University. Hangzhou, China. anzhang@hdu.edu.cn

‡Correspondence author.

§Department of Computing Science, University of Alberta. Edmonton, Alberta T6G 2E8, Canada. {rchin, acai2, guohui}@ualberta.ca

1 Introduction

Wang et al. [21] consider a multiprocessors rescheduling problem, which combines two important sub-fields of extensively studied scheduling research, namely, scheduling with rejection and rescheduling.

In this rescheduling problem, a set of jobs has been planned in an original schedule to minimize their total completion time on a number of parallel identical machines, assuming but not with 100% certainty their availability at the beginning of the planned schedule. The need for rescheduling arises due to some jobs could not arrive in time, but delayed to a certain time point; the decision-maker has to adjust the original schedule to account for the delayed jobs without causing excessive time disruption to the original schedule and to minimize their operational cost. However, it is known that possibly no feasible solution satisfying the rescheduling constraints even exists, and sometimes it is already NP-hard to determine the existence of such a feasible solution [11, 13, 28]. Wang et al. [21] propose to give the decision-maker more freedom, in that they can choose to reject any of the delayed or non-delayed jobs, by paying the corresponding rejection cost, but respecting given upper bounds on the total rejection cost of the rejected jobs and the completion time deviation of each accepted job in the adjusted schedule, and they should seek to minimize their overall operational cost. Specifically, the overall operational cost in [21] is the total completion time of the accepted jobs. Wang et al. [21] presented for the problem a pseudo-polynomial time dynamic programming exact algorithm and a mixed integer linear programming formulation.

In this paper, for the same rescheduling environment as the above, we focus on the time disruption measured as the maximum tardiness of the accepted jobs in the adjusted schedule. Besides respecting given upper bounds on the total rejection cost of the rejected jobs and the tardiness of each accepted job, we study a novel objective function to integrate all three cost components: the total completion time of the accepted jobs, the total rejection cost of the rejected jobs, and the penalty on the maximum tardiness for the accepted jobs. This novel rescheduling problem generalizes several classic NP-hard scheduling problems and we study it from approximation algorithm perspective.

We motivate the rescheduling model and our objective function by a practical hospital application according to Ballestín et al. [1], to build a surgery schedule for elective patients in a hospital. An *elective* patient is one whose surgery can be planned well in advance. For each such patient, we know in advance their operation time and profit loss if rejected (or cost of transference to another hospital). The decision-maker of the surgery unit produces a tentative schedule for all the patients for an established planning period, given the goals sought by the hospital and the available patient information at that moment. Coming to the actual surgery time, the decision-maker constructs the final schedule with the option to reject some patients to deal with the disruptions such as the patient unavailability (and of course, newly arrived patients, if any). In this practice, the hospital needs to carefully consider which patients can be rejected and which accepted operations can be tardy, and converts all these as cost components into the overall operational cost. We may view a patient as a job and an operating room as a machine. Note that in general a hospital owns a fixed number of operating rooms.

In many other modern manufacturing production and service systems, it is also 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 percent of their domestic flights were delayed and 0.3 percent 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 original schedule to account for these disruptions without causing excessive changes to the original schedule. Note that we 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 original schedule, but not a completely new scheduling discarding all prior planning efforts.

Rescheduling has garnered much attention from both important practical issues and interesting theoretical models, and has been studied continuously for nearly three decades. Domains of applications range from automobile industry [3], Space Shuttle ground processing coordination [29], shipyard portal crane access control [5], CrewSolver decision-support system [26], and deregulated power markets [6], to name a few. In terms of rescheduling models, Wu et al. [23] considered the shop rescheduling to minimize the makespan and the total start processing time deviation. Most past research are on a single machine: Unal et al. [20] investigated a rescheduling problem to insert new jobs into an original schedule, to minimize the total weighted completion time (or the makespan) of the new jobs while incurring no additional setup time or tardy jobs; Hall and Potts [10] and Hall et al. [9], and later Zhao et al. [28], also studied the rescheduling problem for inserting a new set (or multiple new sets) of jobs, subject to a limit on the disruption caused to the original schedule or to include the cost of disruption into the objective function; Hall and Potts [11] considered a rescheduling model to respond to job delays, subject to a limit on allowable disruption to any job, and for minimizing the total weighted completion time, they proposed an exact algorithm, a linear-time approximation algorithm, and a fully polynomial time approximation scheme (FPTAS); Liu and Ro [13] studied another rescheduling problem to respond to machine unavailability, and the goal is to minimize the makespan (or maximum lateness) under a maximum allowable time deviation; Liu et al. [15] and Luo et al. [15] extended the study of Liu and Ro [13] to minimize the total weighted completion time; Yin et al. [25] extended the study of Liu and Ro [13] to multiple parallel identical machines, with the disruption limited to one machine unavailability.

In the above rescheduling models, the decision-maker is asked to adjust the schedule to account for disruptions, subject to various constraints and optimization goals. However, the decision-maker is not allowed to reject any jobs. The idea of scheduling with rejection was first introduced by Bartal et al. [2], who studied the multiprocessors scheduling problem to minimize the makespan of the accepted jobs plus the total rejection cost. Other studies on scheduling with rejection concerning the total (weighted) completion time in various scheduling environments include Cheng and Sun [4], Li and Lu [12], Manavizadeh et al. [16], Ou and Zhong [17], Shabtay [19], Yin et al. [24], and Zhang et al. [27], among others. The above scheduling with rejection works focus on the deterministic case, i.e., not rescheduling to respond to various disruption.

In fact, in the literature there aren't many works on rescheduling with rejection. Besides the work by Wang et al. [21] to minimize the total completion time, Wang et al. [22] investigated and proposed a genetic algorithm for a bi-objective rejection-allowed rescheduling on a single machine, in response to continuous arrival of new jobs, under the assumption that the jobs can be rejected and the job processing time is a linear function of its start processing time and its amount of

allocated resource, where the two objectives are the operational cost and the schedule deviation cost; Luo et al. [14] considered the rejection-allowed rescheduling on a single machine, in response to a set of delayed jobs, aiming to minimize a similar objective function as ours while keeping the total rejection cost and the maximum tardiness bounded by given thresholds. Luo et al. [14] presented a pseudo-polynomial time dynamic programming exact algorithm and transferred it into an FPTAS when the total rejection cost is unbounded; they left an open question on whether their algorithm design techniques can be extended to the rejection-allowed multiprocessors rescheduling problem studied by Wang et al. [21].

In this paper, we continue the study on rejection-allowed multiprocessors rescheduling problem to respond to job delays [21]. We focus on the time disruption measured as the maximum tardiness of the accepted jobs in the adjusted schedule. Besides respecting given upper bounds on the total rejection cost of the rejected jobs and the tardiness of each accepted job, we study the objective function to also include them as two cost components, together with the main cost component the total completion time. Recall that on a fixed number of parallel identical machines, an optimal original schedule can be planned in polynomial time, in which the jobs are firstly sorted in the shortest processing time (SPT) order and then assigned to the machines sequentially. (That is, the j-th job is assigned to the i-th machine, where $i = (j-1)\%m+1$ and m is the number of identical machines.) The decision-maker adjusts the schedule to accept a subset of jobs, either delayed or non-delayed, for processing, where for calculating the tardiness of an accepted job, its completion time in the original schedule is taken as its virtual due date. This way, our rescheduling model differs from the work of Hall and Potts [11] and Wang et al. [21] to not penalize job earlier completion, and extends it to allow for, but not unlimitedly, job rejection.

We organize the rest of the paper as follows. In Section 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 multiprocessors rescheduling problem in Section 4, and then develop it into a fully polynomial time approximation scheme when the tardiness is unbounded. We present the implementation details and the numerical experiments in Section 5, with discussions. We conclude the paper in Section 6, and suggest some possible future research.

2 Problem definition

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

Let $\mathcal{J} = \{1, 2, \ldots, n\}$ denote the set of jobs each to be processed non-preemptively on one of m parallel identical machines, and let p_j and e_j denote the processing time and the rejection cost of the job j, respectively, for $j = 1, 2, \ldots, n$. We assume p_j and e_j are non-negative integers. All these jobs are accepted for processing in the planned schedule, denoted as π^*, and they are assumed, though not with 100% certainty, available when the planned processing begins (i.e., at time point

zero). In π^*, the job processing order is the SPT order (in particular, jobs $i, i+m, \ldots, i+(n/m)m$[1] are processed on the machine i), which achieves the minimum total job completion 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,

$$p_1 \leq p_2 \leq \ldots \leq p_n. \tag{1}$$

All the jobs will be processed as early as possible in π^*, and their completion times satisfy

$$C_1(\pi^*) \leq C_2(\pi^*) \leq \ldots \leq C_n(\pi^*). \tag{2}$$

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

After the schedule π^* is planned but before it is executed[2], the decision-maker is informed that a subset $\mathcal{D} \subseteq \mathcal{J}$ of jobs will not be available for processing at time point zero until some later time point $r_{\mathcal{D}} > 0$, which creates a disruption to the original schedule π^*. To reduce the negative impact of the delayed jobs and to achieve an acceptable service level, the decision-maker has the option to reject some delayed or non-delayed[3] jobs from the rescheduling, by paying the rejection cost for each rejected job for outsourcing and/or for loss in revenue and customer goodwill. Nevertheless, often they cannot reject jobs unlimitedly, but to keep the total rejection cost within a given bound h; that is, if we let $\mathcal{R}$ denote the subset of rejected jobs by the decision-maker, then

$$\sum_{j \in \mathcal{R}} e_j \leq h. \tag{3}$$

Let $\mathcal{A} = \mathcal{J} \setminus \mathcal{R}$ denote the subset of jobs accepted into rescheduling, and let σ denote a feasible schedule for the accepted jobs. We define the following quantities for each job $j \in \mathcal{A}$:

- $C_j(\pi^*)$: the completion time of job j in π^*, taken as the virtual due date for job j;
- $C_j(\sigma)$: the completion time of job j in σ;
- $T_j(\sigma) = \max\{C_j(\sigma) - C_j(\pi^*), 0\}$: the tardiness of job j.

When there is no ambiguity on which schedule we are referring to, we simplify $C_j(\sigma)$ and $T_j(\sigma)$ to C_j and T_j, respectively, for each job $j \in \mathcal{A}$.

Let $T_{\max} = \max_{j \in \mathcal{A}} T_j$ denote the maximum tardiness in σ, which must be bounded by a given threshold k set to guarantee a reasonable level of service for the accepted jobs. That is,

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

Besides the above two strict upper bounds on the total rejection cost and the maximum tardiness, respectively, the decision-maker wishes to minimize their the total operational cost for rescheduling, consisting of three components: the total job completion time, the total rejection cost[4], and the

[1]Here n/m is the quotient of the integer division and, if $i > n\%m$ then the job $i + (n/m)m$ is void.

[2]A standard treatment can be applied if the disruption is informed after the processing of available jobs has started, see [21].

[3]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 kept for processing.

[4]The total rejection cost component does not have to be scaled since we may always scale the individual rejection costs at the front.

maximum tardiness. To wrap into a single objective, we set up a non-negative constant scaling factor μ for the maximum tardiness.

We write our rescheduling problem in the three-field notation $\alpha \mid \beta \mid \gamma$ [8]. In this notation, α is the scheduling environment and $\alpha = Pm$ indicates a constant number m of parallel identical machines; β describes the job characteristics or restrictive requirements, in which we use "rej" for rejection-allowed, "$r_{\mathcal{D}}$" for a subset $\mathcal{D}$ of jobs delayed to time point $r_{\mathcal{D}}$, "$\sum_{j\in\mathcal{R}} e_j \le h$" for the total rejection cost being bounded by h, and "$T_{\max} \le k$" for the maximum tardiness being bounded by k; and γ defines the objective function(s) to be minimized, which is a single objective $\sum_{j\in\mathcal{A}} C_j + \sum_{j\in\mathcal{R}} e_j + \mu T_{\max}$. That is, our multiprocessors rescheduling problem is

$$Pm \mid \text{rej}, r_{\mathcal{D}}, \sum_{j\in\mathcal{R}} e_j \le h, T_{\max} \le k \mid \sum_{j\in\mathcal{A}} C_j + \sum_{j\in\mathcal{R}} e_j + \mu T_{\max}, \tag{5}$$

which is NP-hard, since the special and classical case $1 \mid r_{\mathcal{D}} \mid \sum C_j$, that is, single machine scheduling with one nonzero release date, is already binary/weakly NP-hard [18] (by setting $e_j = 1$ for all $j \in \mathcal{J}$ and $h = 0$, $k = +\infty$ and $\mu = 0$ in Problem (5)).

3 Structural properties

In this section, we prove a number of structural properties on some optimal schedules of the accepted jobs for Problem (5). Later we design a dynamic programming exact algorithm to compute such an optimal schedule.

Recall that for Problem (5), we assume the jobs are given in the SPT order (in Eq. (1)), with an overhead of $O(n \log n)$ time; the original optimal schedule π^* plans to process the jobs in this order and the job completion times satisfy Eq. (2). We refer a job by its index in the SPT order.

Let σ^* denote an optimal reschedule by the decision-maker who accepts the subset $\mathcal{A}$ of jobs for processing. We remind the reader that not only a delayed job of $\mathcal{D}$ can be rejected, but also a non-delayed job of $\mathcal{J} \setminus \mathcal{D}$ if the decision-maker wishes. We use the time point $r_{\mathcal{D}}$ to partition the schedule σ^* into the *earlier* and the *later* schedules. Namely, the prefix of the schedule σ^* in which the jobs are started processing strictly before time point $r_{\mathcal{D}}$ is the earlier schedule; the remainder of the schedule σ^* is the later schedule. One clearly sees that the accepted delayed jobs are all processed in the later schedule. Specifically, we also use the earlier (the later, respectively) schedule on the machine i to refer to the earlier (the later, respectively) sub-schedule of σ^* on the machine i. The next lemma summarizes the desired structural properties of σ^*.

Lemma 1 *For Problem (5), there exists an optimal schedule σ^* 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 feasible schedule for the accepted jobs.

To prove item (b), we notice that the jobs in the earlier schedule on a machine in σ^* could be processed by different machines in π^*, but they are all available at time 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 σ', 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,

$$C_i(\sigma') + C_j(\sigma') = (s + p_i) + (s + p_i + p_j) \leq (s + p_j) + (s + p_i + p_j) = C_i(\sigma^*) + C_j(\sigma^*);$$

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

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 σ^* 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 σ' is

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

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. □

4 Algorithmic results

In this section, we first present a pseudo-polynomial time exact algorithm using dynamic programming, and then develop the exact algorithm into a fully polynomial time approximation scheme when the maximum tardiness is unbounded, using the standard sparsing technique.

For convenience, denote $[j] = \{0, 1, 2, \ldots, j\}$ for every non-negative integer j. We remark that sometimes the element 0 does not have its meaning but serves as the boundary condition.

4.1 A dynamic programming exact algorithm

Our exact algorithm uses dynamic programming based on the structural properties of the target optimal schedule stated in Lemma 1. Essentially we only enumerate all the possible combinations of sub-schedules on the m machines, when a sub-schedule on a machine is defined by the start processing time in the later schedule, the total processing time of the earlier schedule, the total processing time of the later schedule, the total completion time, the maximum tardiness, and the total rejection cost. What is not recorded in dynamic programming is the detailed individual job information on whether it is accepted for processing, and if accepted which one of the earlier and the later schedules it is processed in.

Recall that the jobs of $\mathcal{J}$ are given in the SPT order in Eq. (1) with an overhead of $O(n \log n)$ time, and that their virtual dues $C_j(\pi^*)$ are computed in $O(n)$ time. Consider the job subset containing the first j jobs in the SPT order, that is, $[j] = \{0, 1, 2, \ldots, j\}$[5]; we consider only those feasible schedules to process some of $[j]$, in which (see Lemma 1) the jobs are processed as early as possible, the jobs in each earlier schedule are processed in their SPT order, and the jobs in each later schedule are processed in their SPT order. We called these feasible schedules *the partial schedules* on the job subset $[j]$.

Given a combination of the start processing times in the later schedules on the m machines, namely $(s_1, s_2, \ldots, s_m)$ such that $s_1 \geq s_2 \geq \ldots \geq s_m \geq r_{\mathcal{D}}$ (since these m machines are identical), we use a $(2m+4)$-dimensional vector $(j; M_1, M_2, \ldots, M_m, P_1, P_2, \ldots, P_m, Z, T, R)$ 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 $j = 0, 1, \ldots, n-1$. Let $p_{\max} = \max_{j \in \mathcal{J}} p_j$. Then we only need to examine those s_i's such that

$$r_{\mathcal{D}} - 1 + p_{\max} \geq s_1 \geq s_2 \geq \ldots \geq s_m \geq r_{\mathcal{D}}, \tag{6}$$

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

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

We show below that, in general, a true state $(j; M_1, M_2, \ldots, M_m, P_1, P_2, \ldots, P_m, Z, T, R)$ with $0 \leq j < n$ leads to no more than $2m+1$ true states of the form $(j+1; M'_1, M'_2, \ldots, M'_m, P'_1, P'_2, \ldots, P'_m, Z', T', R')$, 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 $(j+1; M'_1, M'_2, \ldots, M'_m, P'_1, P'_2, \ldots, P'_m, Z', T', R')$, we distinguish the following two cases on job $j+1$ being delayed or not:

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

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 an earlier schedule or a later schedule due to the SPT order. Appending to the earlier schedule on machine i requires $M_i < r_{\mathcal{D}}$, $M_i + p_{j+1} \leq s_i$, and its tardiness $M_i + p_{j+1} - C_{j+1}(\pi^*) \leq k$; 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:

- $(j+1; M_1, M_2, \ldots, M_m, P_1, P_2, \ldots, P_m, Z, T, R + e_{j+1})$, if $R + e_{j+1} \leq h$;

[5] Again, here 0 serves as the boundary condition; for example, [0] represents the empty set.

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

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

Similar to Case 1, in this case, job $j+1$ can be rejected if $R + e_{j+1} \le h$. 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 $s_i + P_i + p_{j+1} - C_{j+1}(\pi^*) \le k$. In summary, there are at most $m+1$ true states:

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

At the end, we need to double check each (tentatively) true state of the form $(n; M_1, M_2, \ldots, M_m, P_1, P_2, \ldots, P_m, Z, T, R)$, and keep it if $s_i = \max\{M_i, r_{\mathcal{D}}\}$ for every $i \in [m]$. Then, we calculate its value for the objective function as $Z + R + \mu T$. By a standard backtracking from a state achieving the minimum value, one can obtain an optimal schedule 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 bound the number of states used by the dynamic programming algorithm as follows:

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

where we assume without loss of generality that $r_{\mathcal{D}} \le n p_{\max}$ and $h \le \sum_{j \in \mathcal{J}} e_j$. Therefore, the number of states is in $O(n^{m+3}(r_{\mathcal{D}} + p_{\max})^m p_{\max}^{2m+1} kh)$, 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 $O((2m+1)n^{m+3}(r_{\mathcal{D}} + p_{\max})^m p_{\max}^{2m+1} kh)$-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 completion times, included in Table 1.

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

The five jobs assigned to machine 1 in π^*					The five jobs assigned to machine 2 in π^*				
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

From the allowed maximum tardiness $T_{\max} \le k = 15$, 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 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).

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 5 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 a better implementation of the dynamic programming algorithm (see Section 5), and the objective function 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.

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	$(0; 0, 0, 0, 0, 0, 0, 0)$		
2	$(1; 0, 0, 0, 0, 0, 0, 12)$	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	

No.	True State	Source State	Comments/Objective Value
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	
.....			
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 $(10; 63, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot)$ corresponding to six feasible reschedules under the constraint $(s_1, s_2) = (63, 50)$, respectively, and the minimum objective function 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.

Through the tracing, we learn the following technical details for implementation:

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 Table 1); such annotations are effective in reducing the number of true states. We note that annotations may be made for a subset of jobs, but developing rules is probably too much work.
2. The combination in which $s_i = r_{\mathcal{D}}$ for each machine i is likely the best starting point, and the achieved minimum objective function value can be used for pruning states associated with the other combinations. For the illustrating instance, the minimum objective function value

for the combination $(50, 50)$ is 278, which actually prunes away states 104 and 173, among others.

3. For an $s_i > r_{\mathcal{D}}$, if there is no subset of jobs (which can be assigned to earlier schedules) of total processing time equal to s_i, then no combination containing such an s_i should be examined. Perhaps run one time dynamic programming algorithm for Subset-Sum to determine the possible s_i values; any saving is worthwhile.
4. For two true states $S^1 = (j; M_1, M_2, \ldots, M_m, P_1, P_2, \ldots, P_m, Z^1, T^1, R^1)$ and $S^2 = (j; M_1, M_2, \ldots, M_m, P_1, P_2, \ldots, P_m, Z^2, T^2, R^2)$, if $Z^1 \leq Z^2$, $T^1 \leq T^2$ and $R^1 \leq R^2$ then we say S^1 is better than S^2 and denote as $S^1 \preceq S^2$; in this case, state S^2 can be eliminated from further consideration. We note that in order to compare, all the existing true states on the job subset $[j]$ will have to be sorted in some way, such as the dictionary order; however, sorting and/or maintaining the sorted order could cost too much time.

4.3 A fully polynomial time approximation scheme

Recall that our multiprocessors 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 one nonzero release date, is already binary/weakly NP-hard [18] (by setting $e_j = 1$ for all $j \in \mathcal{J}$ and $h = 0$, $k = +\infty$ and $\mu = 0$ in Problem (5)).

From Lemma 1, 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 assigns the subset of accepted jobs to each of the m earlier schedules and the m later schedules. These decisions determine the total rejection cost and the job processing order in each of the earlier schedules and the later schedules, and henceforth determine the maximum job tardiness $T_{\max}$ and the start processing time of each later schedule.

The NP-hardness of $1 \mid r_{\mathcal{D}} \mid \sum C_j$ proven in [18] is by a reduction from the KNAPSACK/SUBSET-SUM problem [7], 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 [18] *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 Section 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, 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 $(s_1, s_2, \ldots, s_m)$, satisfying Eq. (6). In the following approximation scheme, however, we use a different collection $\mathcal{S}$ of combinations.

Given a small positive ϵ, let,

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

Define the collection of combinations

$$\mathcal{S} = \{(s_1, s_2, \ldots, s_m) \mid s_1 \ge s_2 \ge \ldots \ge s_m, s_i \in I^s, 1 \le i \le m\}; \tag{9}$$

note that $|\mathcal{S}| \le (v_0+1)^m \in O(4^m n^m (\log M)^m/\epsilon^m)$, where $M = \max\{r_{\mathcal{D}}, p_{\max}\}$. In the following approximation scheme, for a combination $(s_1, s_2, \ldots, s_m) \in \mathcal{S}$, a state $(j; M_1, M_2, \ldots, M_m, P_1, P_2, \ldots, P_m, Z, T, R)$ 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; (We note that the constraint "$T \le k$" is removed.)
(v) the total rejection cost is exactly R and $R \le h$;
(vi) and for every $i \in [m]$, $M_i \le s_i$. (We note that the constraint "if $j = n$ then $\max\{M_i, r_{\mathcal{D}}\} = s_i$" is removed.)

Additionally, from $r_{\mathcal{D}} \le s_i \le \delta^n(r_{\mathcal{D}} - 1 + p_{\max}) \le (1+\epsilon)(r_{\mathcal{D}} - 1 + p_{\max})$, 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 function value. We nevertheless leave the partial schedule as it is (as our target is only near optimality).

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

(1) $s_i \le s'_i < \delta s_i$ for $1 \le i \le m$,

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

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

For the combination $(s'_1, s'_2, \ldots, s'_m)$, 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 σ associated with the true state $(n; M_1, M_2, \ldots, M_m, P_1, P_2, \ldots, P_m, Z, T, R)$, one sees that we achieve a new reschedule σ' which differs from σ only in the start processing times of the later schedules.

Assume a job j is assigned in the later schedule of σ' on machine i and let C'_j denote its completion time; let C_j denote its completion time in σ. 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 $(\delta - 1)s_i \le (\delta - 1)(r_{\mathcal{D}} - 1 + p_{\max})$. We thus conclude that there is a true state $(n; M_1, M_2, \ldots, M_m, P_1, P_2, \ldots, P_m, Z', T', R)$ with $Z' < \delta Z$ and $T' < T + (\delta - 1)(r_{\mathcal{D}} - 1 + p_{\max})$. □

For any given combination $(s_1, s_2, \ldots, s_m) \in \mathcal{S}$, 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 size into intervals of geometrically increasing lengths, resulting in only polynomially many intervals in 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 $(j; M_1, M_2, \ldots, M_m, P_1, P_2, \ldots, P_m, Z, T, R)$, where the range of each variable is stated in Eq. (7). We thus see that all but the first dimensions have pseudo-polynomial sizes, 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 sizes as

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

and let

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

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

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

Given a $(2m+3)$-dimensional vector

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

a grid $G(j; i_\ell, 1 \leq \ell \leq 2m+3)$ is defined as the collection of all the states $(j; M_1, M_2, \ldots, M_m, P_1, P_2, \ldots, P_m, Z, T, R)$, 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 grid is *true* if at least one of its states is true, and one such true state will be selected as the *representative* of the grid. The initial true state $(0; 0, 0, \ldots, 0)$ is always picked as the representative for the grid $G(0; 0, 0, \ldots, 0)$.

The main task in the second step is to determine which grids are true and for each true grid to decide its representative. This is done in the similar fashion as the exact algorithm. Note that the initial grid $G(0; 0, 0, \ldots, 0)$ is true.

Given a general true grid $G(j; i_\ell, 1 \le \ell \le 2m+3)$, let $(j; M_1, M_2, \ldots, M_m, P_1, P_2, \ldots, P_m, Z, T, R)$ 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 $(j+1; M'_1, M'_2, \ldots, M'_m, P'_1, P'_2, \ldots, P'_m, Z', T', R')$ and thus they enable up to $2m+1$ grids of the form $G(j+1; i'_\ell, 1 \le \ell \le 2m+3)$ to be true. We always pick the true state with the minimum total rejection cost as the representative for a true grid $G(j+1; i'_\ell, 1 \le \ell \le 2m+3)$. In more details, when a true state $(j+1; M'_1, M'_2, \ldots, M'_m, P'_1, P'_2, \ldots, P'_m, Z', T', R')$ is newly identified and it belongs to the grid $G(j+1; i'_\ell, 1 \le \ell \le 2m+3)$, 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 $(j+1; M''_1, M''_2, \ldots, M''_m, P''_1, P''_2, \ldots, P''_m, Z'', T'', R'')$ such that $R'' \le R'$, then the representative remains unchanged; if the grid had been previously identified as true with its representative $(j+1; M''_1, M''_2, \ldots, M''_m, P''_1, P''_2, \ldots, P''_m, Z'', T'', R'')$ such that $R'' > R'$, then the state $(j+1; M'_1, M'_2, \ldots, M'_m, P'_1, P'_2, \ldots, P'_m, Z', T', R')$ replaces to be the representative. As side information, the source grid *leading to* the grid $G(j+1; i'_\ell, 1 \le \ell \le 2m+3)$ needs to be updated properly, to be used in the backtracking.

In the last step, the algorithm examines all true grids of the form $G(n; i_\ell, 1 \le \ell \le 2m+3)$, and for each calculates the value of the objective function for its representative state $(n; M_1, M_2, \ldots, M_m, P_1, P_2, \ldots, P_m, Z, T, R)$ as $Z+R+\mu T$. The same as in the exact algorithm, by a standard backtracking from a state achieving the minimum value, denoted as $(n; M^\epsilon_1, M^\epsilon_2, \ldots, M^\epsilon_m, P^\epsilon_1, P^\epsilon_2, \ldots, P^\epsilon_m, Z^\epsilon, T^\epsilon, R^\epsilon)$, 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 μ to penalize a large maximum tardiness.

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 $M = \max\{n, r_{\mathcal{D}}, p_{\max}, h\}$.*

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

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

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

sentative, satisfying

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

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

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

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

I.e., the statement holds for the state $(j+1; M_1, M_2, \ldots, M_m, P_1, P_2, \ldots, P_m, Z, T, R+e_{j+1})$ too.[6]

If job $j+1$ can be accepted into the earlier schedule on machine i, that is, $M_i + p_{j+1} \le s_i$, then the state $(j+1; M_1, \ldots, M_{i-1}, M_i+p_{j+1}, M_{i+1}, \ldots, M_m, P_1, P_2, \ldots, P_m, Z+M_i+p_{j+1}, \max\{T, M_i + p_{j+1} - C_{j+1}(\pi^*)\}, R)$ is true for the combination $(s_1, s_2, \ldots, s_m)$. For the picked representative state $(j; M'_1, M'_2, \ldots, M'_m, P'_1, P'_2, \ldots, P'_m, Z', T', R')$, 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 $(j+1; M'_1, \ldots, M'_{i-1}, M'_i + p_{j+1}, M'_{i+1}, \ldots, M'_m, P'_1, P'_2, \ldots, P'_m, Z' + M'_i + p_{j+1}, \max\{T', M'_i + p_{j+1} - C_{j+1}(\pi^*)\}, R')$ is evaluated by the algorithm $A(\epsilon)$ to be true for the combination $(s'_1, \ldots, s'_{i-1}, s''_i, s'_{i+1}, \ldots, s'_m) \in \mathcal{S}$.[7] For the true grid to which this state belongs, the picked representative state is denoted as $(j+1; M''_1, M''_2, \ldots, M''_m, P''_1, P''_2, \ldots, P''_m, Z'', T'', R'')$

[6] In Case 1, no new combination of $\mathcal{S}$ is involved.

[7] In Case 2, a possibly new combination $(s'_1, \ldots, s'_{i-1}, s''_i, s'_{i+1}, \ldots, s'_m) \in \mathcal{S}$ is involved.

and satisfies

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

Using

$$M_i' + p_{j+1} \leq \delta^j M_i + p_{j+1} < \delta^j (M_i + p_{j+1}),$$
$$Z' + M_i' + p_{j+1} \leq \delta^j Z + \delta^j M_i + p_{j+1} < \delta^j (Z + M_i + p_{j+1}),$$

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

we have

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

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

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

If job $j+1$ can be accepted into the later schedule on machine i, then the state $(j+1; M_1, M_2, \ldots, M_m, P_1, \ldots, P_{i-1}, P_i + p_{j+1}, P_{i+1}, \ldots, P_m, Z + s_i + P_i + p_{j+1}, \max\{T, s_i + P_i + p_{j+1} - C_{j+1}(\pi^*)\}, R)$ is

true for the combination $(s_1, s_2, \ldots, s_m)$. For the picked representative state $(j; M'_1, M'_2, \ldots, M'_m, P'_1, P'_2, \ldots, P'_m, Z', T', R')$, one sees that the state $(j+1; M'_1, M'_2, \ldots, M'_m, P'_1, \ldots, P'_{i-1}, P'_i + p_{j+1}, P'_{i+1}, \ldots, P'_m, Z' + s'_i + P'_i + p_{j+1}, \max\{T', s'_i + P'_i + p_{j+1} - C_{j+1}(\pi^*)\}, R')$ is evaluated by the algorithm $A(\epsilon)$ to be true for the combination $(s'_1, s'_2, \ldots, s'_m) \in \mathcal{S}$.[8] For the true grid to which this state belongs, the picked representative state is denoted as $(j+1; M''_1, M''_2, \ldots, M''_m, P''_1, P''_2, \ldots, P''_m, Z'', T'', R'')$ and satisfies

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

Using

$$P'_i + p_{j+1} \le \delta^j P_i + p_{j+1} < \delta^j (P_i + p_{j+1}),$$

$$Z' + s'_i + P'_i + p_{j+1} \le \delta^j Z + \delta^j s_i + \delta^j P_i + p_{j+1} < \delta^j (Z + s_i + P_i + p_{j+1}),$$

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

we have

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

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

[8] In Case 3, no new combination of $\mathcal{S}$ is involved.

that is, the statement holds for the state $(j+1; M_1, M_2, \ldots, M_m, P_1, \ldots, P_{i-1}, P_i + p_{j+1}, P_{i+1}, \ldots, P_m, Z + s_i + P_i + p_{j+1}, \max\{T, s_i + P_i + p_{j+1} - C_{j+1}(\pi^*)\}, R)$. 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 $(n, M_1, M_2, \ldots, M_m, P_1, P_2, \ldots, P_m, Z, T, R)$ is true for the combination $(s_1, s_2, \ldots, s_m)$ of start processing times by the exact dynamic programming, with its objective function 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 $(n, M'_1, M'_2, \ldots, M'_m, P'_1, P'_2, \ldots, P'_m, Z', T', R')$ for some combination $(s'_1, s'_2, \ldots, s'_m) \in \mathcal{S}$ as a representative with its total rejection cost $R' \leq R \leq h$ and its objective function 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 ϵ to be $\frac{\epsilon}{1+\mu}$, the algorithm $A(\epsilon)$ is a $(1+\epsilon)$-approximation.

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 $M = \max\{n, r_{\mathcal{D}}, p_{\max}, h\}$. □

5 Numerical experiments

5.1 Instance simulation

Wang et al. [21] have implemented their algorithms and performed numerical experiments. For ease of comparison we use the same data simulation scheme, 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 $n \in \{4, 6, 8, 10, 11, 12, ..., 20, 30, 40, 50\}$ 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.[9] We ran our experiments on a 16GB memory;[10] if our C program run into "SEGFAULT", then it indicates the memory is exhausted. We set the running time limit on each instance at 2 hours, and force termination when time is up.

5.2 Results

Table 3: Performance of the dynamic programming exact algorithm: average running time over 20 randomly generated instances for each pair of (m, n). Entries with "time" or "memory" are those on which the C program didn't terminate within 2 hours time limit or crashed out of memory of 16GB, respectively.

n	$m = 2$	3	4	6	8	10
4				-	-	-
6					-	-
8						-
10						
12						
14						
16						
18						
20						
30						
40						
50						

[11]

5.3 Discussion

Wang et al. [21] showed very poor performance for the dynamic programming exact algorithm. However, our numerical experiments demonstrated the outstanding performance. Our dynamic programming algorithm is even more powerful than the one in [21], that is, ours can also produce an optimal reschedule that minimizes the total completion time. The outstanding performance comes from the careful memory allocation in our implementation, which requests for storage for only those true states.

[9] Or 100?

[10] Need detailed specification of the computer(s)? Rylan can we use the CSC 159 machines?

[11] We likely plot them in figures.

Table 4: Performance of the FPTAS: for $\epsilon \in \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}\}$, average running time over 20 randomly generated instances and average quality of the produced solutions compared against the optimal solutions by the dynamic programming exact algorithm. Entries with "time" or "memory" are those on which the C program didn't terminate within 2 hours time limit or crashed out of memory of 16GB, respectively.

n	$m = 2$	3	4	6	8	10
10						
20						
30						
40						
50						

6 Conclusions

In this paper, we addressed partially an open problem left in [14] to examine the rejection-allowed multiprocessors rescheduling problem to respond to job delays, with the objective to minimize the sum of total completion time of the accepted jobs, the total rejection cost of the rejected jobs, and the penalty on the maximum tardiness, subject to an upper bounded total rejection cost and an upper bounded maximum tardiness. We observed several important structural properties associated with an optimal reschedule, and fully employed them into the design of a pseudo-polynomial time dynamic programming exact algorithm. When the maximum tardiness is unbounded, we then non-trivially used the sparsing technique to further develop the exact algorithm into a fully polynomial time approximation scheme. We thus achieve the best possible approximation algorithm when the maximum tardiness is unbounded, and confirm partially a claim in [14] when the number of parallel identical machines is fixed.

Note that the pseudo-polynomial time dynamic programming exact algorithm works for the more general setting in which both the total rejection cost and the maximum tardiness are upper bounded; but in this more general setting, development of an FPTAS seems challenging, for the reason that bounding the maximum tardiness and the total rejection cost simultaneously cannot be done within the sparsing technique. It is therefore interesting to see whether the dynamic programming exact algorithm can lead to an FPTAS (or perhaps a PTAS) for this more general setting. If the answer is negative, then can we design an FPTAS or a PTAS when the maximum tardiness is bounded but the total rejection cost is unbounded? which is the case for the similar single machine rescheduling problem [14].

Declarations of interest:

None.

References

[1] F. Ballestín, Á. Pérez, and S. Quintanilla. Scheduling and rescheduling elective patients in operating rooms to minimise the percentage of tardy patients. *Journal of Scheduling*, 22:107–118, 2019.

[2] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, and L. Stougie. Multiprocessor scheduling with rejection. In *Proceedings of ACM-IEEE SODA 2000*, pages 95–103.

[3] J. C. Bean, J. R. Birge, J. Mittenthal, and C. E. Noon. Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research*, 39:470–483, 1991.

[4] Y. Cheng and S. Sun. Scheduling linear deteriorating jobs with rejection on a single machine. *European Journal of Operational Research*, 194:18–27, 2007.

[5] J. Clausen, J. Larsen, A. Larsen, and J. Hansen. Disruption management – operations research between planning and execution. *OR/MS Today*, 28:40–43, 2001.

[6] K. Dahal, K. Al-Arfaj, and K. Paudyal. Modelling generator maintenance scheduling costs in deregulated power markets. *European Journal of Operational Research*, 240:551–561, 2015.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, San Francisco, 1979.

[8] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annuals of Discrete Mathematics*, 5:287–326, 1979.

[9] N. G. Hall, Z. Liu, and C. N. Potts. Rescheduling for multiple new orders. *INFORMS Journal on Computing*, 19:633–645, 2007.

[10] N. G. Hall and C. N. Potts. Rescheduling for new orders. *Operations Research*, 52:440–453, 2004.

[11] N. G. Hall and C. N. Potts. Rescheduling for job unavailability. *Operations Research*, 58:746–755, 2010.

[12] D. Li and X. Lu. Two-agent parallel-machine scheduling with rejection. *Theoretical Computer Science*, 703:66–75, 2017.

[13] Z. Liu and Y. K. Ro. Rescheduling for machine disruption to minimize makespan and maximum lateness. *Journal of Scheduling*, 17:339–352, 2014.

[14] W. Luo, M. Jin, B. Su, and G. Lin. An approximation scheme for rejection-allowed single-machine rescheduling. *Computers & Industrial Engineering*. `https://doi.org/10.1016/j.cie.2020.106574`.

[15] W. Luo, T. Luo, R. Goebel, and G. Lin. Rescheduling due to machine disruption to minimize the total weighted completion time. *Journal of Scheduling*, 21:565–578, 2018.

[16] N. Manavizadeh, A. H. Goodarzi, M. Rabbani, and F. Jolai. Order acceptance/rejection policies in determining the sequence in mixed model assembly lines. *Applied Mathematical Modelling*, 37:2531–2351, 2013.

[17] J. Ou and X. Zhong. Bicriteria order acceptance and scheduling with consideration of fill rate. *European Journal of Operational Research*, 262:904–907, 2017.

[18] A. H. G. Rinnooy Kan. *Machine scheduling problems: classification, complexity, and computations*. Springer, 1976.

[19] D. Shabtay. The single machine serial batch scheduling problem with rejection to minimize total completion time and total rejection cost. *European Journal of Operational Research*, 233:64–74, 2014.

[20] A. T. Unal, R. Uzsoy, and A. S. Kiran. Rescheduling on a single machine with part-type dependent setup times and deadlines. *Annals of Operations Research*, 70:93–113, 1997.

[21] D. Wang, Y. Yin, and T. C. E. Cheng. Parallel-machine rescheduling with job unavailability and rejection. *Omega*, 81:246–260, 2018.

[22] D.-J. Wang, F. Liu, and Y. Jin. A multi-objective evolutionary algorithm guided by directed search for dynamic scheduling. *Computers & Operations Research*, 79:279–290, 2017.

[23] S. D. Wu, R. H. Storer, and P. C. Chang. A rescheduling procedure for manufacturing systems under random disruptions. In G. Fandel, T. Gulledge, and A. Jones, editors, *New Directions for Operations Research in Manufacturing*, pages 292–306. Springer, Berlin, Heidelberg, 1992.

[24] Y. Yin, T. C. E. Cheng, D. Wang, and C. C. Wu. Improved algorithms for single-machine serial-batch scheduling with rejection to minimize total completion time and total rejection cost. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46:1578–1588, 2016.

[25] Y. Yin, T. C. E. Cheng, and D.-J. Wang. Rescheduling on identical parallel machines with machine disruptions to minimize total completion time. *European Journal of Operational Research*, 252:737–749, 2016.

[26] G. Yu, M. Argüello, G. Song, S. M. McCowan, and A. White. A new era for crew recovery at continental airlines. *INFORMS Journal on Applied Analytics*, 33:5–22, 2003.

[27] L. Zhang, L. Lu, and J. Yuan. Single-machine scheduling under the job rejection constraint. *Theoretical Computer Science*, 411:1877–1882, 2010.

[28] Q. Zhao, L. Lu, and J. Yuan. Rescheduling with new orders and general maximum allowable time disruptions. *4OR*, 14:261–280, 2016.

[29] M. Zweben, E. Davis, B. Daun, and M. J. Deale. Scheduling and rescheduling with iterative repair. *IEEE Transactions on Systems, Man, and Cybernetics*, 23:1588–1596, 1993.

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 5 lists all the transitions from a true state with j to a true 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 a better implementation of the dynamic programming algorithm (see Section 5), and the objective function 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.

Table 5: 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	$(0; 0, 0, 0, 0, 0, 0, 0)$		
2	$(1; 0, 0, 0, 0, 0, 0, 12)$	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	$\preceq$ 16; State 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	
26	$(6; 0, 12, 0, 0, 17, 3, 80)$	14, Case 2	
27	$(6; 14, 0, 0, 0, 23, 5, 68)$	15, Case 2	

No.	True State	Source State	Comments/Objective Value
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	
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	

No.	True State	Source State	Comments/Objective Value
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	$\preceq$ 92; State 92 eliminated
99	$(9; 5, 35, 35, 30, 225, 13, 80)$	59, Case 1	
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	$\succeq$ 100; State 107 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	

No.	True State	Source State	Comments/Objective Value
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	
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	

No.	True State	Source State	Comments/Objective Value
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	
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 $(10; 63, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot)$ corresponding to six feasible reschedules under the constraint $(s_1, s_2) = (63, 50)$, respectively, and the minimum objective function 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.