Journal of Scheduling (2018) 21:565–578
https://doi.org/10.1007/s10951-018-0575-z

Rescheduling due to machine disruption to minimize the total weighted completion time

Wenchang Luo[1,2] · Taibo Luo[2,3] · Randy Goebel[2] · Guohui Lin[2]

Published online: 3 July 2018

Abstract

We investigate a single machine rescheduling problem that arises from an unexpected machine unavailability, after the given set of jobs has already been scheduled to minimize the total weighted completion time. Such a disruption is represented as an unavailable time interval and is revealed to the production planner before any job is processed; the production planner wishes to reschedule the jobs to minimize the alteration to the originally planned schedule, which is measured as the maximum time deviation between the original and the new schedules for all the jobs. The objective function in this rescheduling problem is to minimize the sum of the total weighted completion time and the weighted maximum time deviation, under the constraint that the maximum time deviation is bounded above by a given value. That is, the maximum time deviation is taken both as a constraint and as part of the objective function. We present a pseudo-polynomial time exact algorithm and a fully polynomial time approximation scheme.

Keywords Rescheduling · Machine disruption · Total weighted completion time · Approximation scheme

1 Introduction

In most modern production industries and service systems, various kinds of disruptions will occur, such as order cancellations, new order arrivals, machine breakdown, and labor or material shortages. An ideal scheduling system is expected to effectively adjust an originally planned schedule to account for such disruptions, in order to minimize the effects of the disruption on overall performance. The extent of an alteration to the originally planned schedule, to be minimized, becomes either a second objective function (e.g., to model measurable costs), or is formulated as a constraint to model hard-to-estimate costs, which may be incorporated into the original objective function.

In this paper, we investigate the single machine scheduling problem with the objective to minimize the total weighted completion time. Rescheduling arises because of unexpected machine unavailability, which we represent as an unavailable time interval. This unavailability is revealed to the production planner after the given set of jobs has already been scheduled but processing has not begun. The production planner wishes to reschedule the jobs to minimize the alteration to the originally planned schedule, measured as the maximum time deviation between the original and the new schedules for all jobs. The maximum time deviation is taken both as a constraint and as part of the objective function; that is, the maximum time deviation is bounded above by a given threshold value, and the new objective function becomes to minimize the sum of the total weighted completion time and the weighted maximum time deviation.

This work was partially supported by NSERC Canada, NSF China and CSC China.

✉ Guohui Lin
guohui@ualberta.ca

Wenchang Luo
wenchang@ualberta.ca

Taibo Luo
taibo@ualberta.ca

Randy Goebel
rgoebel@ualberta.ca

1 Faculty of Science, Ningbo University, Ningbo 315211, Zhejiang, China

2 Department of Computing Science, University of Alberta, Edmonton, AB T6G 2E8, Canada

3 School of Economics and Management, Xidian University, Xi'an 710126, China

1.1 Problem description and definitions

We formally present our rescheduling problem in what follows, including definitions and notation to be used throughout the paper.

We are given a set of jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$, where the job J_j has an integer weight w_j and requires an integer *non-preemptive* processing time p_j on a single machine, with the original objective to minimize the *total weighted completion time*. This problem is denoted as $(1 \mid\mid \sum_{j=1}^{n} w_j C_j)$ under the three-field classification scheme (Graham et al. 1979), where C_j denotes the completion time of the job J_j. It is known that the *weighted shortest processing time* (WSPT) rule gives an optimal schedule for the problem $(1 \mid\mid \sum_{j=1}^{n} w_j C_j)$. We thus assume that the jobs are already sorted in the WSPT order, that is, $\frac{p_1}{w_1} \le \frac{p_2}{w_2} \le \ldots \le \frac{p_n}{w_n}$, and we denote this order/schedule as π^*, referred to as the *original* schedule (also called the *pre-planned* schedule, or *pre-schedule*, in the literature).

Once obtaining the original schedule π^*, the production planner has the completion time of the job J_j denoted as $C_j(\pi^*)$ and thus sets the delivery time at $C_j(\pi^*)$ for the finished job J_j, or otherwise within the time window $[C_j(\pi^*) - k, C_j(\pi^*) + k]$ for some given flexibility threshold $k \ge 0$. This promised delivery time is taken seriously as a hard constraint meaning that slight adjustments to the original schedule, due to various reasons, are allowed only if the new completion time of the job is within the window.

The rescheduling arises due to a machine disruption: the machine becomes unavailable in the time interval $[T_1, T_2]$, where $0 \le T_1 < T_2$. We assume, without loss of generality, that this information is known to us at time zero, so no job is yet processed. (Otherwise, one may remove those processed jobs from consideration, and for the partially processed job, either run it to completion and then remove it, or stop processing it immediately.)

Let σ be a schedule after resolving the machine disruption. That is, no job of $\mathcal{J}$ is processed in the time interval $[T_1, T_2]$ in σ. As in the existing literature, we use the following notation: for each job J_j,

$S_j(\sigma)$: the starting time of the job J_j in the schedule σ;
$C_j(\sigma)$: the completion time of the job J_j in the schedule σ, and thus $C_j(\sigma) = S_j(\sigma) + p_j$;
$C_j(\pi^*)$: the completion time of the job J_j in the original schedule π^*;
$\Delta_j(\pi^*, \sigma) = |C_j(\sigma) - C_j(\pi^*)|$: the *time deviation* of the job J_j in the two schedules.

Let $\Delta_{\max}(\pi^*, \sigma) \triangleq \max_{j=1}^{n}\{\Delta_j(\pi^*, \sigma)\}$ denote the *maximum time deviation* for all jobs. When it is clear from the context, we simplify the terms $S_j(\sigma)$, $C_j(\sigma)$, $\Delta_j(\pi^*, \sigma)$ and $\Delta_{\max}(\pi^*, \sigma)$ to S_j, C_j, Δ_j and $\Delta_{\max}$, respectively.

The time deviation of a job measures how much its actual completion time is off the originally planned, and thus it can model the penalties resulted from the delivery time change to the satisfaction of the customers. Such varying penalties could be difficult for the producer to quantify, and therefore in this paper the maximum time deviation is taken as a hard constraint, that is, $\Delta_{\max} \le k$ for the given flexibility threshold k. On the other hand, from the production perspective, the time deviation of a detailed job does not, but mostly the maximum time deviation of all jobs would increase the internal production cost associated with the rescheduling of resources. Therefore, we add the maximum time deviation to the objective function, balanced with a suitable factor $\mu \ge 0$. That is, the goal of rescheduling is to minimize the sum of the weighted maximum time deviation and the total weighted completion time, i.e., $\mu\Delta_{\max} + \sum_{j=1}^{n} w_j C_j$. Our problem is denoted as $(1, h_1 \mid \Delta_{\max} \le k \mid \mu\Delta_{\max} + \sum_{j=1}^{n} w_j C_j)$ under the three-field classification scheme (Graham et al. 1979), where the first field "$1, h_1$" denotes a single machine with a single unavailable time period, the second field "$\Delta_{\max} \le k$" indicates the hard constraint on the maximum time deviation, and the last field is the objective function.

We remark that in the literature, reviewed below, there are works on the objective to minimize the sum of the *makespan* and the weighted maximum time deviation, with a bounded maximum time deviation the same as ours (Liu and Ro 2014), and works on the objective to minimize the sum of the total completion time and the weighted *total* time deviation, but the maximum time deviation is unbounded (Qi et al. 2006). All these different rescheduling constraints and objectives are driven by different real applications.

1.2 Related research

We next review major research on the variants of the rescheduling problem, inspired by many practical applications. To name a few such applications, Bean et al. (1991) investigated an automobile industry application, and proposed a heuristic *match-up* scheduling approach to accommodate disruptions from multiple sources. Zweben et al. (1993) studied the GERRY scheduling and rescheduling system that supports Space Shuttle ground processing, using a heuristic constraint-based iterative repair method. Clausen et al. (2001) considered a shipyard application, where the goal for rescheduling is to store large steel plates for efficient access by two non-crossing portal cranes that move the plates to appropriate places. Vieira et al. (2003), Aytug et al. (2005), Herroelen and Leus (2005) and Yang et al. (2005) provided extensive reviews of the rescheduling literature, including taxonomies, strategies and algorithms, for both deterministic and stochastic environments.

In a seminal paper on rescheduling theory for a single machine, Hall and Potts (2004b) considered the rescheduling problem required to deal with the arrival of a new set of jobs, which disrupts the pre-planned schedule of the original jobs. Such a problem is motivated by the unexpected arrival of new orders in practical manufacturing systems. First, the set of original jobs has been optimally scheduled to minimize a cost function, typically the maximum lateness or the total completion time; but no job has yet been executed. In this case, promises have been made to the customers based on the original schedule. Then an unexpected new set of jobs arrives before the processing starts; the production planner needs to insert the new jobs into the existing schedule seeking to minimize change to the original plan. The measure of change to the original schedule is the maximum or total sequence deviation, or the maximum or total time deviation. For both cases—where the measure of change is modeled only as a constraint, or where the measure of change is modeled both as a constraint and is added to the original cost objective—the authors provide either an efficient algorithm or an intractability proof for several problem variants.

Yuan and Mu (2007) studied a rescheduling problem similar to the one in Hall and Potts (2004b), but with the objective to minimize the makespan subject to a limit on the maximum sequence deviation of the original jobs; they show that such a solution is polynomial time solvable. Hall et al. (2007) considered an extension of the rescheduling problem in Hall and Potts (2004b), where the arrivals of multiple new sets of jobs create repeated disruptions to minimize the maximum lateness of the jobs, subject to a limit on the maximum time deviation of the original jobs; they proved the NP-hardness and presented several approximation algorithms with their worst-case performance analysis.

Hall and Potts (2004a) also studied the case where the disruption is a delayed subset of jobs (or called *job unavailability*), with the objective to minimize the total weighted completion time, under a limit on the maximum time deviation; they presented an exact algorithm, an intractability proof, a constant-ratio approximation algorithm, and a fully polynomial time approximation scheme (FPTAS). Hoogeveen et al. (2012) studied the case where the disruption is the arrival of new jobs and the machine needs a setup time to switch between processing an original job and processing a new job; their bi-criterion objective is to minimize the makespan and to minimize the maximum (or total) positional deviation or the maximum (or total) time deviation, with certain assumptions on the setup times. They presented a number of polynomial time exact algorithms and intractability proofs for several problem variants. Zhao and Yuan (2013) examined the case where the disruption is the arrival of new jobs which are associated with release dates, formulated a bi-objective function to minimize the makespan and to minimize the total sequence deviation, under a limit on the total sequence deviation; they presented a strongly polynomial time algorithm for finding all Pareto optimal points of the problem. Wang et al. (2017) also discussed a bi-objective single machine scheduling problem with continuous arrival of new jobs (which nonetheless can be rejected), and proposed a dynamic evolutionary multi-objective optimization algorithm incorporating a directed search strategy. Wang et al. (2015) proposed a knowledge-based multi-objective evolutionary algorithm for the case where the machine breakdown is stochastic and extra resources are available to match up the original schedule, and the objective is to minimize the sum of the total completion time and the extra resource consumption cost to match up the original schedule.

Qi et al. (2006) and Liu and Ro (2014) considered the same machine disruption as ours—the machine is unavailable for a period of time. In Qi et al. (2006), the objective is to minimize the weighted sum of the total completion time and the total time deviation, without any constraint on the time deviation; in Liu and Ro (2014), the objective is to minimize the sum of the makespan (or the maximum job lateness) and the weighted maximum time deviation, and with a given upper bound on the maximum time deviation. Qi et al. (2006) presented only a heuristic for the problem, but a 3.5-approximation if the total time deviation in the objective is replaced by the total time earliness. Liu and Ro (2014) presented a pseudo-polynomial time exact algorithm, a 2-approximation algorithm, and an FPTAS. Yin et al. (2016) studies the rescheduling problem on multiple identical parallel machines with multiple machine disruptions, and the bi-criterion objective is to minimize the total completion time and to minimize the total virtual tardiness (or the maximum time deviation); in addition to hardness results, they presented a two-dimensional FPTAS when there is exactly one machine disruption.

Among all related research in the above, the work of Qi et al. (2006) and Liu and Ro (2014) is the most relevant to our work in terms of the scheduling environment, and the work of Hall and Potts (2004b, a) is the most relevant in terms of the original objective function. After the first version of our work (Luo et al. 2017) was under review, we noticed a recent online published article (Liu et al. 2017), where the problem $(1, h_1 \mid \Delta_{\max} \le k \mid \sum_{j=1}^{n} w_j C_j)$, i.e., the special case of our problem with $\mu = 0$, is studied, and the authors presented an $O(n^2 T_1)$-time exact algorithm and also claimed an $O(n^3 \log W/\epsilon)$-time $(1+\epsilon)$-approximation algorithm[1] with W being the total weighted completion time of a feasible schedule, for any $\epsilon > 0$.

1.3 Our contributions and organization

Our problem $(1, h_1 \mid \Delta_{\max} \le k \mid \mu\Delta_{\max} + \sum_{j=1}^{n} w_j C_j)$ includes three interesting special cases, some of which have

[1] There is an error in the $(1+\epsilon)$-approximation algorithm, for which the claimed running time $O(n^3 \log W/\epsilon)$ is flawed (Liu and Lin 2017).

received attention in the literature: when the given bound k is sufficiently large, the time deviation constraint becomes void and our problem reduces to the total cost problem $(1, h_1 \mid\mid \mu\Delta_{\max} + \sum_{j=1}^{n} w_j C_j)$; when the time deviation factor $\mu = 0$, our problem reduces to the constrained rescheduling problem $(1, h_1 \mid \Delta_{\max} \le k \mid \sum_{j=1}^{n} w_j C_j)$, without the need to minimize the time deviation (Liu et al. 2017); and finally, when the given bound k is sufficiently large and the time deviation factor $\mu = 0$, our problem reduces to the classic scheduling problem with a machine unavailability period $(1, h_1 \mid\mid \sum_{j=1}^{n} w_j C_j)$ (Lee 1996), which is NP-hard.

The rest of the paper is organized as follows: In Sect. 2, we derive structural properties that are associated with the optimal solutions to our rescheduling problem $(1, h_1 \mid \Delta_{\max} \le k \mid \mu\Delta_{\max} + \sum_{j=1}^{n} w_j C_j)$. In Sect. 3, we present an $O(n^3 T_1)$-time exact algorithm DP- 1, which is pseudo-polynomial. In Sect. 4, we first develop an $O(n^2 T_1)$-time exact algorithm DP- 2 solving the special case where $\mu = 0$, this is, the problem $(1, h_1 \mid \Delta_{\max} \le k \mid \sum_{j=1}^{n} w_j C_j)$;[2] this leads to an $O(n^2 T_1 k)$-time exact algorithm for the general problem $(1, h_1 \mid \Delta_{\max} \le k \mid \mu\Delta_{\max} + \sum_{j=1}^{n} w_j C_j)$, which is also pseudo-polynomial but slower than DP- 1 as often $n < k$. Based on the algorithm DP- 2, we first present an $O(n^5 \log T_1 \log P \log W/\epsilon^3)$-time $(1+\epsilon)$-approximation algorithm for $(1, h_1 \mid \Delta_{\max} \le k \mid \sum_{j=1}^{n} w_j C_j)$, for any $\epsilon > 0$; by calling this $(1+\epsilon)$-approximation algorithm $O(\log k/\epsilon)$ times, we present an $O(n^5 \log T_1 \log P \log W \log k/\epsilon^4)$-time $(1 + \epsilon)$-approximation algorithm for $(1, h_1 \mid \Delta_{\max} \le k \mid \mu\Delta_{\max} + \sum_{j=1}^{n} w_j C_j)$, for any $\epsilon > 0$, where n is the number of jobs, the machine is unavailable $[T_1, T_2]$, P is the total job processing time, and W is the total weighted completion time of any feasible schedule. That is, we have an FPTAS for the problem $(1, h_1 \mid \Delta_{\max} \le k \mid \mu\Delta_{\max} + \sum_{j=1}^{n} w_j C_j)$.

We remark that one major contribution in our FPTAS for the problem $(1, h_1 \mid \Delta_{\max} \le k \mid \mu\Delta_{\max} + \sum_{j=1}^{n} w_j C_j)$ is the guaranteed $O(\log k/\epsilon)$ calls to the $(1+\epsilon)$-approximation algorithm for $(1, h_1 \mid \Delta_{\max} \le k \mid \sum_{j=1}^{n} w_j C_j)$. To see this, one might propose to call the $(1 + \epsilon)$-approximation algorithm for $(1, h_1 \mid \Delta_{\max} \le i \mid \sum_{j=1}^{n} w_j C_j)$ for each $i = 0, 1, \ldots, k$, followed by picking the value of i that minimizes the quantity $\mu i + \sum_{j=1}^{n} w_j C_j$. Such an algorithm does find a solution with the objective value within $1 + \epsilon$ of the optimum; however, its total running time is $O(kn^5 \log T_1 \log P \log W/\epsilon^3)$, which is pseudo-polynomial.

We conclude our paper in the last section, with some final remarks.

[2] We remark that our algorithm differs from the exact algorithm in Liu et al. (2017), though both of $O(n^2 T_1)$ time.

2 Preliminaries

Firstly, from the NP-hardness of the classic scheduling problem with a machine unavailability period $(1, h_1 \mid\mid \sum_{j=1}^{n} w_j C_j)$ (Lee 1996), we conclude that our rescheduling problem $(1, h_1 \mid \Delta_{\max} \le k \mid \mu\Delta_{\max} + \sum_{j=1}^{n} w_j C_j)$ is also NP-hard.

Recall that we are given a set of jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ in the WSPT order, where each job J_j has a positive weight w_j and a positive processing time p_j, a machine unavailability period $[T_1, T_2]$ with $0 \le T_1 < T_2$, an upper bound k on the maximum time deviation, and a balancing factor $\mu \ge 0$. All these w_j's, p_j's, T_1, T_2, k are integers, and μ is a rational number. For any feasible schedule σ to the rescheduling problem, from $\Delta_{\max} \le k$ we conclude that $C_j(\pi^*) - k \le C_j(\sigma) \le C_j(\pi^*) + k$ for every job J_j. For ease of presentation, we partition σ into two halves, similar to the existing literature: the prefix of the schedule σ with the jobs completed before or at time T_1 is referred to as the *earlier schedule* of σ, and the suffix of the schedule σ with the jobs completed after time T_2 is referred to as the *later schedule* of σ. We assume, without loss of generality, that with the same $\Delta_{\max}$, all the jobs are processed as early as possible in σ (to achieve the minimum possible total weighted completion time). Let σ^* denote an optimal schedule to the rescheduling problem.

2.1 Problem setting

Let

$$p_{\min} \triangleq \min_{j=1}^{n} p_j, \quad P_i \triangleq \sum_{j=1}^{i} p_j, \quad \text{and} \quad P \triangleq P_n. \tag{1}$$

Using the original schedule $\pi^* = (1, 2, \ldots, n)$, we compute

$$j_1 \triangleq \min\{j \mid C_j(\pi^*) > T_1\}, \quad j_2 \triangleq \min\{j \mid S_j(\pi^*) \ge T_2\}, \tag{2}$$

i.e., J_{j_1} is the first job in π^* completed strictly after time T_1 and J_{j_2} is the first job in π^* starting processing at or after time T_2. One clearly sees that if j_1 is void, i.e., no job is completed strictly after time T_1, then no rescheduling is necessary; in the sequel, we always assume that the job J_{j_1} exists. Nevertheless, we note that j_2 could be void, which means that all the jobs start processing strictly before time T_2 in π^*.

We may furthermore assume the following relations hold among $p_{\min}$, P, T_1, T_2 and k, to ensure that the rescheduling problem is non-trivial:

$$p_{\min} \le T_1 < P, \text{ and } T_2 - S_{j_1}(\pi^*) \le k. \quad (3)$$

For a quick proof, firstly, if $T_1 < p_{\min}$, then no job can be processed before the machine unavailability period and thus the schedule π^* remains optimal except that the job processing starts at time T_2 instead of time 0; secondly, if $T_1 \ge P$, then no rescheduling is necessary. Lastly, from the definition of the job J_{j_1} in Eq. (2), we conclude that at least one job among $J_1, J_2, \ldots, J_{j_1}$ must be completed after time T_2 in any feasible rescheduling solution, with its time deviation at least $T_2 - S_{j_1}(\pi^*)$. Therefore, $T_2 - S_{j_1}(\pi^*) \le k$, as otherwise no feasible solution exists.

2.2 Structure properties of the optimal schedules

There is a very regular property of our target optimal schedules to the rescheduling problem, stated in the following lemma, of which the proof is done by a repeated job swapping process with the details in "Appendix A".

Lemma 2.1 *There exists an optimal schedule σ^* for the rescheduling problem $(1, h_1 \mid \Delta_{\max} \le k \mid \mu\Delta_{\max} + \sum_{j=1}^{n} w_j C_j)$, in which (a) the jobs in the earlier schedule are in the same order as they appear in π^*; and (b) the jobs in the later schedule are also in the same order as they appear in π^*.*

There are several more properties listed in the next Lemma 2.2, which are important to the design and analysis of the algorithms to be presented. Most of these properties also hold for the optimal schedules to a similar makespan rescheduling problem $(1, h_1 \mid \Delta_{\max} \le k \mid \mu\Delta_{\max} + C_{\max})$ (Liu and Ro 2014). We remark that the makespan is only a part of our objective, and the original schedule for the makespan scheduling problem $(1, h_1 \mid \Delta_{\max} \le k \mid \mu\Delta_{\max} + C_{\max})$ can be arbitrary. However, for our problem, the jobs in π^* are in the WSPT order. Our goal is to compute an optimal schedule satisfying (the properties stated in) Lemmas 2.1 and 2.2, and thus we examine only those feasible schedules σ satisfying Lemmas 2.1 and 2.2.

Recall that we assume, for every feasible schedule σ to the rescheduling problem with the same maximum time deviation $\Delta_{\max}$, all the jobs are processed as early as possible in σ (to achieve the minimum possible total weighted completion time). However, this does not rule out the possibility that the machine would idle, due to the unavailable period $[T_1, T_2]$ and/or the constraint on the maximum time deviation $\Delta_{\max}$. For example, the machine has to idle for a period of time right before time T_1, if no job can be fitted into this period for processing; also, the machine might choose to idle for a period of time and then proceeds to process a job, in order to obtain a smaller $\Delta_{\max}$. In the sequel, our discussion is about the latter kind of machine idling. The good news is that there are optimal schedules in which the machine has at most one such idle time period, as shown in the next Lemma 2.2. This makes our search for optimal schedules much easier.

Lemma 2.2 *There exists an optimal schedule σ^* for the rescheduling problem $(1, h_1 \mid \Delta_{\max} \le k \mid \mu\Delta_{\max} + \sum_{j=1}^{n} w_j C_j)$, in which (a) $C_j(\sigma^*) \le C_j(\pi^*)$ for each job J_j in the earlier schedule; (b) the machine idles for at most one period of time in the earlier schedule; (c) each job in the earlier schedule after the idle time period is processed exactly $\Delta_{\max}$ time units earlier than in π^*; (d) the jobs in the earlier schedule after the idle time period are consecutive in π^*; (e) the job in the earlier schedule right after the idle time period has a starting time at or after time T_2 in π^*; (f) the machine does not idle in the later schedule; and (g) the first job in the later schedule reaches the maximum time deviation among all the jobs in the later schedule.*

Proof Item (a) is a direct consequence of Lemma 2.1, since the jobs in the earlier schedule are in the same order as they appear in π^*, which is the WSPT order.

Proofs of (b)–(g) are similar to those in Liu and Ro (2014), where they are proven for an arbitrary original schedule, while the WSPT order is only a special order. For completeness, the proofs are included in Appendix B. □

Among the jobs $J_1, J_2, \ldots, J_{j_1}$, we know that some of them will be processed in the later schedule of the optimal schedule σ^*. By Lemma 2.1, we conclude that the first job in the later schedule is from $J_1, J_2, \ldots, J_{j_1}$. We use J_a to denote this job, and consequently $(J_1, J_2, \ldots, J_{a-1})$ remains as the prefix of the earlier schedule. The time deviation of J_a is $\Delta_a = T_2 - S_a(\pi^*) \le k$ (which could be used to further narrow down the candidates for J_a).

Corollary 1 *In an optimal schedule σ^* satisfying Lemmas 2.1 and 2.2, suppose the job J_a is the first job in the later schedule. For each $j = a+1, a+2, \ldots, j_2 - 1$, if the job J_j is in the earlier schedule, then its time deviation Δ_j is less than Δ_a.*

Proof From the definition of j_2 in Eq. (2), we know that $S_j(\pi^*) < T_2$, and therefore its time deviation is $\Delta_j < T_2 - S_a(\pi^*) = \Delta_a$. □

3 A dynamic programming exact algorithm

In this section, we develop an exact algorithm DP-1 for the rescheduling problem $(1, h_1 \mid \Delta_{\max} \le k \mid \mu\Delta_{\max} + \sum_{j=1}^{n} w_j C_j)$, to compute an optimal schedule σ^* satisfying Lemmas 2.1 and 2.2. The key idea is as follows: By Lemma 2.2(g), we first guess the job J_a that starts processing at time T_2 in σ^*, with its time deviation $\Delta_a = T_2 - S_a(\pi^*) \le$

k. From this initial partial schedule,[3] our algorithm constructs some feasible full schedules satisfying Lemmas 2.1 and 2.2. To guarantee that our algorithm constructs an optimal full schedule in pseudo-polynomial time, we use a "hash" function to map a partial schedule to a triple, such that for each triple only the partial schedule achieving the minimum total weighted completion time of the jobs is saved for the computation, to be described in detail in the following. At the end, the full schedule with the minimum objective function value is returned as the solution σ^*.

We notice from Lemmas 2.1 and 2.2(a) that the time deviations of the jobs in the earlier schedule of σ^* are non-decreasing, with the first $(a-1)$ ones being 0's. From the constraint $\Delta_{\max} \le k$, the last job in the schedule π^* that can possibly be in the earlier schedule of σ^* is J_{j_3} with

$$j_3 \triangleq \max\{j \mid C_j(\pi^*) - k \le T_1\}. \tag{4}$$

The exact algorithm DP- 1 is a dynamic programming, that sequentially assigns the job J_{j+1} to the partial schedules on the first j jobs $J_1, J_2, \dots, J_j$; each such partial schedule is described (hashed) as a triple $(j; \ell_j, e_j)$, where $a \le j \le j_3$, no machine idling period in the earlier schedule, ℓ_j is the total processing time of the jobs in the earlier schedule, and e_j is the index of the last job in the earlier schedule. Clearly, with the same ℓ_j and e_j, all the partial schedules mapping to the triple $(j; \ell_j, e_j)$ have the *same* maximum time deviation $\max\{\Delta_{e_j}, \Delta_a\}$; among them, the *minimum* total weighted completion time of the jobs is denoted as $Z(j; \ell_j, e_j)$, and the partial schedule achieving this minimum is saved for (called *associated with*) the triple and used for the subsequent computation.

For ease of presentation, we partition the jobs into four subsequences:

$$\begin{aligned} \mathcal{J}_1 &= (J_1, J_2, \dots, J_{a-1}),\\ \mathcal{J}_2 &= (J_{a+1}, J_{a+2}, \dots, J_{j_2-1}),\\ \mathcal{J}_3 &= (J_{j_2}, J_{j_2+1}, \dots, J_{j_3}),\\ \mathcal{J}_4 &= (J_{j_3+1}, J_{j_3+2}, \dots, J_n). \end{aligned} \tag{5}$$

From Lemma 2.1 and Eq. (4), we know that $\mathcal{J}_1$ is a prefix of the earlier schedule and $\mathcal{J}_4$ is a suffix of the later schedule, and thus DP- 1 takes care of only the jobs of $\mathcal{J}_2 \cup \mathcal{J}_3$. Let $Z(\mathcal{J}_1)$ denote the total weighted completion time of the jobs in $\mathcal{J}_1$, and $Z(\mathcal{J}_4)$ denote the total weighted completion time of the jobs in $\mathcal{J}_4$ by starting the job processing at time 0.

[3] In the entire paper, we examine only feasible partial schedules for the prefixes of the WSPT job sequence $\pi^* = (J_1, J_2, \dots, J_n)$; these partial schedules can always be completed into feasible full schedules.

The (only) starting partial schedule for DP- 1 is described as

$$\begin{aligned}(a; \ell_a, e_a) &= (a; P_{a-1}, a-1) \text{ and } Z(a; P_{a-1}, a-1)\\ &= Z(\mathcal{J}_1) + w_a(T_2 + p_a),\end{aligned} \tag{6}$$

in which the job J_a starts processing at time T_2. In general, given a triple $(j; \ell_j, e_j)$ with $a \le j < j_3$, that is associated with a partial schedule on the first j jobs of the total weighted completion time of the first j jobs $Z(j; \ell_j, e_j)$, DP- 1 assigns the next job J_{j+1} of $\mathcal{J}_2 \cup \mathcal{J}_3$ as follows

(1) to generate at most three new partial schedules each described as a triple $(j+1; \ell_{j+1}, e_{j+1})$,
(2) to compute the total weighted completion time of the first $j+1$ jobs using $Z(j; \ell_j, e_j)$, and
(3) if the total is strictly smaller, to update $Z(j+1; \ell_{j+1}, e_{j+1})$ and correspondingly to update the saved partial schedule for the triple $(j+1; \ell_{j+1}, e_{j+1})$;
(4) if a non-empty machine idling period is inserted before J_{j+1} in the earlier schedule of the new partial schedule, then the partial schedule is directly completed optimally to a full schedule using Lemma 2.2(d).

Case 1 J_{j+1} is added in the later schedule to obtain a partial schedule described as:

$$\begin{aligned}&(j+1; \ell_{j+1}, e_{j+1}) = (j+1; \ell_j, e_j) \text{ and } Z(j+1; \ell_j, e_j)\\ &\quad = Z(j; \ell_j, e_j) + w_{j+1}(T_2 + P_{j+1} - \ell_j),\end{aligned} \tag{7}$$

in which the completion time of the job J_{j+1} is $C_{j+1} = T_2 + P_{j+1} - \ell_j$. The feasibility holds since $\Delta_{j+1} \le \Delta_a$ by Lemma 2.2(g).

Case 2 If J_{j+1} can fit in the earlier schedule, that is $\ell_j + p_{j+1} \le T_1$ and $C_{j+1}(\pi^*) - (\ell_j + p_{j+1}) \le k$, then we add J_{j+1} in the earlier schedule without inserting a machine idling period to obtain a feasible partial schedule described as:

$$\begin{aligned}&(j+1; \ell_{j+1}, e_{j+1}) = (j+1; \ell_j + p_{j+1}, j+1) \text{ and } Z(j\\ &\quad +1; \ell_j + p_{j+1}, j+1) = Z(j; \ell_j, e_j)\\ &\quad + w_{j+1}(\ell_j + p_{j+1}),\end{aligned} \tag{8}$$

in which the completion time of the job J_{j+1} is $C_{j+1} = \ell_j + p_{j+1}$ and $\Delta_{j+1} = C_{j+1}(\pi^*) - (\ell_j + p_{j+1}) \le k$.

Case 3 If $\ell_j + p_{j+1} < T_1$ and $C_{j+1}(\pi^*) - (\ell_j + p_{j+1}) > \max\{\Delta_{e_j}, \Delta_a\}$, then we add J_{j+1} in the earlier schedule and insert a non-empty machine idling period right before it. The following Lemma 3.1 states that the exact length of the machine idling period can be determined in $O(n-j)$-time, and during the same time, the new partial schedule is directly optimally completed into a full schedule using Lemma 2.2(d), for which the maximum time deviation $\Delta_{\max}$ and the objective function value $\hat{Z}_n$ are also computed.

Lastly, for every triple $(j; \ell_j, e_j)$ with $j = j_3$ and its associated partial schedule on the first j_3 jobs, the algorithm DP- 1 completes the partial schedule by assigning all the jobs of $\mathcal{J}_4$ in the later schedule, starting at time $T_2 + P_j - \ell_j$, to obtain a full schedule described as the triple

$$\begin{aligned}(n; \ell_n, e_n) = (n; \ell_j, e_j) \text{ and } Z(n; \ell_j, e_j) &= Z(j; \ell_j, e_j) + Z(\mathcal{J}_4) \\ &\quad + \left(\sum_{i=j+1}^{n} w_i\right)(T_2 + P_j - \ell_j), \end{aligned} \tag{9}$$

for which the maximum time deviation is $\Delta_{\max} = \max\{\Delta_{e_j}, \Delta_a\}$ and the objective function value is

$$\hat{Z}_n \triangleq \mu \max\{\Delta_{e_j}, \Delta_a\} + Z(n; \ell_j, e_j). \tag{10}$$

Lemma 3.1 *When the job J_{j+1} is added in the earlier schedule of the partial schedule associated with a triple $(j; \ell_j, e_j)$ and a non-empty machine idling period is inserted before it, then*

(a) *the minimum possible length of this period is $\max\{1, C_{j+1}(\pi^*) - (\ell_j + p_{j+1}) - k\}$;*
(b) *the maximum possible length of this period is $\min\{C_{j+1}(\pi^*) - (\ell_j + p_{j+1}) - \max\{\Delta_{e_j}, \Delta_a\}, T_1 - (\ell_j + p_{j+1})\}$;*
(c) *the exact length can be determined in $O(n - j)$-time;*
(d) *during the same time an optimal constrained full schedule is achieved directly, together with its the maximum time deviation $\Delta_{\max}$ and the objective function value $\hat{Z}_n$.*

Proof 1 is a trivial lower bound given that all job processing times are positive integers. Next, if $C_{j+1}(\pi^*) - (\ell_j + p_{j+1}) > k$, then we cannot process J_{j+1} immediately at time ℓ_j as this would result in a time deviation greater than the given upper bound k; and the earliest possible starting time is $C_{j+1}(\pi^*) - p_{j+1} - k$, leaving a machine idling period of length $C_{j+1}(\pi^*) - (\ell_j + p_{j+1}) - k$. This proves item (a).

The maximum possible length for the period is no more than $C_{j+1}(\pi^*) - (\ell_j + p_{j+1}) - \max\{\Delta_{e_j}, \Delta_a\}$, for otherwise Δ_{j+1} would be less than $\max\{\Delta_{e_j}, \Delta_a\}$. However, $\Delta_{j+1} < \Delta_{e_j}$ contradicts Lemma 2.1 and Lemma 2.2(a) that suggest the jobs in the earlier schedule should have non-decreasing time deviations; $\Delta_{j+1} < \Delta_a$ contradicts Lemma 2.2(c) that $\Delta_{\max} \geq \Delta_a$. Also, since J_{j+1} is added to the earlier schedule, the length of the idling period has to be at most $T_1 - (\ell_j + p_{j+1})$. This proves item (b).

Let $L = \max\{1, C_{j+1}(\pi^*) - (\ell_j + p_{j+1}) - k\}$ and $U = \min\{C_{j+1}(\pi^*) - (\ell_j + p_{j+1}) - \max\{\Delta_{e_j}, \Delta_a\}, T_1 - (\ell_j + p_{j+1})\}$. For each value $i \in [L, U]$, we 1) start processing J_{j+1} at time $\ell_j + i$, 2) then continuously process succeeding jobs in the earlier schedule until the one won't fit in, and 3) lastly process all the remaining jobs in the later schedule. This gives a full schedule, denoted as π^i, with the total weighted completion time denoted Z_n^i, and the maximum time deviation $\Delta_{\max}^i = \Delta_{j+1} = C_{j+1}(\pi^*) - (\ell_j + p_{j+1}) - i$. Its objective function value is $\hat{Z}_n^i \triangleq \mu\Delta_{\max}^i + Z_n^i$.

It follows that the interval $[L, U]$ can be partitioned into $O(n - j)$ subintervals, such that for all values in a subinterval say $[i_1, i_2]$, the jobs assigned to the earlier schedule are identical; and consequently the objective function value $\hat{Z}_n^i$ is a linear function in i, where $i_1 \leq i \leq i_2$. It follows that among all these full schedules, the minimum objective function value must be achieved at one of $\hat{Z}_n^{i_1}$ and $\hat{Z}_n^{i_2}$. That is, we in fact do not need to compute the full schedules π^i's with those i's such that $i_1 < i < i_2$, and consequently there are only $O(n - j)$ full schedules to be computed.

These subintervals can be determined as follows: when $i = L$, let the jobs fit into the earlier schedule after the machine idling period be $J_{j+1}, J_{j+2}, \ldots, J_{j+s}$, then the first subinterval is $[L, L + T_1 - C_{j+s}]$, where C_{j+s} is the completion time of J_{j+s} in the full schedule π^i; the second subinterval is $[L + T_1 - C_{j+s} + 1, L + T_1 - C_{j+s-1}]$; the third subinterval is $[L + T_1 - C_{j+s-1} + 1, L + T_1 - C_{j+s-2}]$; and so on, until the last interval hits the upper bound U.

The optimal length of the machine idling period is the one i^* that minimizes $\hat{Z}_n^i$, among the $O(n - j)$ computed full schedules, and we obtain a corresponding constrained optimal full schedule directly from the partial schedule associated with the triple $(j; \ell_j, e_j)$, together with its the maximum time deviation and the objective function value. □

Theorem 3.1 *The algorithm* DP- 1 *solves the rescheduling problem $(1, h_1 \mid \Delta_{\max} \leq k \mid \mu\Delta_{\max} + \sum_{j=1}^{n} w_j C_j)$, with its running time in $O(n^3 T_1)$, where n is the number of jobs and $[T_1, T_2]$ is the machine unavailable time interval.*

Proof In the above, we presented the dynamic programming algorithm DP- 1 to compute a full schedule for each triple $(n; \ell_n, e_n)$ satisfying Lemmas 2.1 and 2.2, under the constraint that the job J_a starts processing at time T_2. The final output schedule is the one with the minimum $\hat{Z}_n$, among all the choices of J_a such that $\Delta_a \leq k$. Its optimality lies in the triple representation for the partial schedules and the recurrences we developed in Eqs. (6–9). There are $O(j_1)$ choices for J_a, and for each J_a we compute a partial schedule for each triple $(j; \ell_j, e_j)$, where $a \leq j \leq j_3$ or $j = n$, $P_{a-1} \leq \ell_j \leq T_1$, and $a - 1 \leq e_j \leq j$. Since one partial schedule leads to at most three other candidate partial schedules, and each takes an $O(1)$-time in average to compute, the overall running time is $O(n^3 T_1)$. □

After the first version of our work (Luo et al. 2017) was under review, we noticed a recent online published article (Liu et al. 2017), where the problem $(1, h_1 \mid \Delta_{\max} \leq k \mid \sum_{j=1}^{n} w_j C_j)$, i.e., the special case of our problem with $\mu = 0$, is studied, and the authors presented an $O(n^2 T_1)$-time exact algorithm. Based on their algorithm, one can derive an $O(n^2 T_1 k)$-time exact algorithm for our problem, through enumerating all possible values for $\Delta_{\max}$. We note that when $n < k$ (also noting that n is linear, while k is exponential, in the size of the instance), our algorithm DP- 1 is faster.

4 An FPTAS

The route to the FPTAS for the rescheduling problem $(1, h_1 \mid \Delta_{\max} \leq k \mid \mu \Delta_{\max} + \sum_{j=1}^{n} w_j C_j)$ is as follows: we first consider the special case where $\mu = 0$, that is $(1, h_1 \mid \Delta_{\max} \leq k \mid \sum_{j=1}^{n} w_j C_j)$, where the maximum time deviation is only upper bounded by k but not taken as part of the objective function, and design another exact algorithm, denoted as DP- 2, using a different dynamic programming recurrence than in DP- 1; then we develop from the algorithm DP- 2 an FPTAS for the special case $\mu = 0$; lastly, we use the FPTAS for the special case polynomial times to design an FPTAS for the general case $\mu \geq 0$.

4.1 Another dynamic programming exact algorithm for $\mu = 0$

In this special case, we have stronger conclusions on the target optimal schedule σ^* than those stated in Lemma 2.2, one of which is that if there is a machine idling period in the earlier schedule of σ^*, then $\Delta_{\max} = k$. This follows from the fact that, if $\Delta_{\max} < k$, we may start processing the jobs after the idling period one-time unit earlier to decrease the total weighted completion time. We conclude this in the following lemma.

Lemma 4.1 *There exists an optimal schedule σ^* for the rescheduling problem $(1, h_1 \mid \Delta_{\max} \leq k \mid \sum_{j=1}^{n} w_j C_j)$, in which if the machine idles in the earlier schedule then $\Delta_{\max} = k$.*

The new exact algorithm DP- 2 heavily relies on this conclusion in Lemma 4.1 (and thus it does not work for the general case). DP- 2 can readily be developed into an FPTAS for $(1, h_1 \mid \Delta_{\max} \leq k \mid \sum_{j=1}^{n} w_j C_j)$.

The framework of the new algorithm DP- 2 is the same, and we continue to use the notations defined in Eqs. (1, 4, 5). But now we use a pair $(j; \ell_j^1)$ (instead of a triple) to describe a partial schedule on the first j jobs, where $a \leq j \leq j_3$, no machine idling period in the earlier schedule and the time deviation of the last job in the earlier schedule is no greater than k, and ℓ_j^1 is the maximum job completion time in the earlier schedule (or equally, the total job processing time in the earlier schedule). Associated with the pair, we let ℓ_j^2 denote the maximum job completion time in the later schedule, which is clearly $\ell_j^2 = T_2 + P_j - \ell_j^1$. Among all the partial schedules mapping to the pair $(j; \ell_j^1)$, the *minimum* total weighted completion time of the jobs is denoted as $Z(j; \ell_j^1)$, and the partial schedule achieving this minimum is saved for (called *associated with*) the pair and used for the subsequent computation.

Starting with guessing J_a (from the pool $\{J_1, J_2, \ldots, J_{j_1}\}$) to be the job started processing at time T_2, such that $\Delta_a = T_2 - S_a(\pi^*) \leq k$, the (only) corresponding partial schedule is described as the pair

$$(a; \ell_a^1) = (a; P_{a-1}) \text{ and } Z(a; P_{a-1}) = Z(\mathcal{J}_1) + w_a(T_2 + p_a). \tag{11}$$

In general, given a pair $(j; \ell_j^1)$ with $a \leq j < j_3$ and its associated partial schedule with the total weighted completion time of the first j jobs $Z(j; \ell_j^1)$, the algorithm DP- 2 assigns the next job J_{j+1} of $\mathcal{J}_2 \cup \mathcal{J}_3$ as follows

(1) to generate at most three new partial schedules each described as a pair $(j+1; \ell_{j+1}^1)$,
(2) to compute the total weighted completion time of the first $j+1$ jobs using $Z(j; \ell_j^1)$, and
(3) if the total is strictly smaller, to update $Z(j+1; \ell_{j+1}^1)$ and correspondingly to update the saved partial schedule for the pair $(j+1; \ell_{j+1}^1)$;
(4) if a non-empty machine idling period is inserted before J_{j+1} in the earlier schedule of the new partial schedule, then the partial schedule is directly completed optimally to a full schedule using Lemma 2.2(d).

Case 1 J_{j+1} is added in the later schedule to obtain a partial schedule described as:

$$\begin{aligned}(j+1; \ell_{j+1}^1) = (j+1; \ell_j^1) \text{ and } Z(j+1; \ell_j^1) \\ = Z(j; \ell_j^1) + w_{j+1}(T_2 + P_{j+1} - \ell_j^1),\end{aligned} \tag{12}$$

in which the completion time of the job J_{j+1} is $C_{j+1} = T_2 + P_{j+1} - \ell_j^1$. The feasibility holds since $\Delta_{j+1} \leq \Delta_a$ by Lemma 2.2(g).

Case 2 If J_{j+1} can fit in the earlier schedule, that is $\ell_j^1 + p_{j+1} \leq T_1$ and $C_{j+1}(\pi^*) - (\ell_j^1 + p_{j+1}) \leq k$, then we add J_{j+1} in the earlier schedule without inserting a machine idling period to obtain a feasible partial schedule described as:

$$(j+1; \ell^1_{j+1}) = (j+1; \ell^1_j + p_{j+1}) \text{ and } Z(j+1; \ell^1_j + p_{j+1}) = Z(j; \ell^1_j) + w_{j+1}(\ell^1_j + p_{j+1}), \tag{13}$$

in which the completion time of the job $C_{j+1} = \ell^1_j + p_{j+1}$ and its time deviation is $\Delta_{j+1} = C_{j+1}(\pi^*) - (\ell^1_j + p_{j+1}) \leq k$.

Case 3 If $\ell^1_j + p_{j+1} < C_{j+1}(\pi^*) - k \leq T_1$, then we add J_{j+1} in the earlier schedule and insert a non-empty machine idling period of length $C_{j+1}(\pi^*) - (\ell^1_j + p_{j+1}) - k$ right before it. (That is, start processing J_{j+1} at time $C_{j+1}(\pi^*) - p_{j+1} - k$.) Then continuously process succeeding jobs in the earlier schedule as long as they fit in (but up to J_{j_3}), and lastly process all the remaining jobs in the later schedule. Assuming the last job fits in the earlier schedule is $J_{j'}$, this gives a full schedule with the maximum time deviation $\Delta_{\max} = k$ and the total weighted completion time

$$Z(j; \ell^1_j) + Z(\{J_{j+1}, \ldots, J_{j'}\}) + \left(\sum_{i=j+1}^{j'} w_i\right)(C_{j+1}(\pi^*) - p_{j+1} - k) + Z(\{J_{j'+1}, \ldots, J_n\}) + \left(\sum_{i=j'+1}^{n} w_i\right)(T_2 + P_j - \ell^1_j),$$

where $Z(\{J_{j+1}, \ldots, J_{j'}\})$ $(Z(\{J_{j'+1}, \ldots, J_n\})$, respectively) denotes the total weighted completion time of the jobs in $\{J_{j+1}, \ldots, J_{j'}\}$ (in $\{J_{j'+1}, \ldots, J_n\}$, respectively) by starting the job processing at time 0.

Lastly, for every pair $(j; \ell^1_j)$ with $j = j_3$ and and its associated partial schedule on the first j_3 jobs, the algorithm DP- 2 completes it by assigning all the jobs of $\mathcal{J}_4$ in the later schedule, starting at time $\ell^2_j = T_2 + P_j - \ell^1_j$, to obtain a full schedule described as

$$(n; \ell^1_n) = (n; \ell^1_j) \text{ and } Z(n; \ell^1_j) = Z(j; \ell^1_j) + Z(\mathcal{J}_4) + \left(\sum_{i=j+1}^{n} w_i\right)(T_2 + P_j - \ell^1_j), \tag{14}$$

for which the maximum time deviation is guaranteed to be no greater than k.

Theorem 4.1 *The algorithm* DP- 2 *solves the rescheduling problem* $(1, h_1 \mid \Delta_{\max} \leq k \mid \sum_{j=1}^n w_j C_j)$ *in* $O(n^2 T_1)$*-time, where* n *is the number of jobs and* $[T_1, T_2]$ *is the machine unavailable time interval.*

Proof In the above, we see that the dynamic programming algorithm DP- 2 computes for each pair $(n; \ell^1_n)$ a full schedule satisfying Lemmas 2.1 and 2.2 and having the minimum total weighted completion time of all the n jobs. These full schedules do not have a machine idling period in the earlier schedule. For each pair $(j; \ell^1_j)$, it might be completed into a full schedule with a machine idling period inserted in the earlier schedule such that its maximum time deviation $\Delta_{\max} = k$. The final output schedule is the one with the minimum total weighted completion time, among all the two kinds of full schedules with all possible choices of J_a such that $\Delta_a \leq k$. Similarly to the proof of Theorem 3.1, the optimality lies in the pair representation for the partial schedules and the recurrences we developed in Eqs. (11–14). There are $O(j_1)$ choices for J_a, and for each J_a we compute all partial schedules described as $(j; \ell^1_j)$, where $a \leq j \leq n$ and $P_{a-1} \leq \ell^1_j \leq T_1$. Since one partial schedule leads to at most three other candidate partial schedules, and each takes on average an $O(1)$-time to compute, the overall running time is $O(n^2 T_1)$. □

We remark that in a recent online published article (Liu et al. 2017), another $O(n^2 T_1)$-time exact algorithm is presented for the problem $(1, h_1 \mid \Delta_{\max} \leq k \mid \sum_{j=1}^n w_j C_j)$.

Corollary 2 *The algorithm* DP- 2 *can be extended to solve the rescheduling problem* $(1, h_1 \mid \Delta_{\max} \leq k \mid \mu\Delta_{\max} + \sum_{j=1}^n w_j C_j)$ *in* $O(n^2 T_1 k)$*-time, where* n *is the number of jobs and* $[T_1, T_2]$ *is the machine unavailable time interval.*

Proof For each $k' = 0, 1, \ldots, k$, we use the algorithm DP- 2 to solve the rescheduling problem $(1, h_1 \mid \Delta_{\max} \leq k' \mid \sum_{j=1}^n w_j C_j)$ in $O(n^2 T_1 k)$-time, and let $W(k')$ denote the achieved minimum total weighted completion time. Then the optimal objective function value for the general rescheduling problem $(1, h_1 \mid \Delta_{\max} \leq k \mid \mu\Delta_{\max} + \sum_{j=1}^n w_j C_j)$ is $\min\{\mu k' + W(k') \mid 0 \leq k' \leq k\}$, which is computed in $O(n^2 T_1 k)$ time. □

4.2 An FPTAS for $\mu = 0$

In this subsection, we convert the exact algorithm DP- 2 in the last subsection into an FPTAS by the sparsing technique. We assume the job J_a is scheduled to start processing at time T_2, and we have a positive real value $\epsilon > 0$. The algorithm is denoted as Approx(a, ϵ), which guarantees to return a feasible schedule such that its total weighted completion time is within $(1 + \epsilon)$ of the constrained optimum, under the constraint that the job J_a is scheduled to start processing at time T_2.

Recall that a basic unit in the exact algorithm DP- 2 in the last subsection is a pair $(j; \ell^1_j)$, of which the associated partial schedule on the first j jobs has the minimum total weighted completion time $Z(j; \ell^1_j)$, there is no machine idle period in the earlier schedule, and the total job processing time in the earlier schedule is ℓ^1_j. We use ℓ^2_j to denote the maximum job completion time in the later schedule, and thus $\ell^2_j = T_2 + P_j - \ell^1_j$, where P_j is the total job processing time of the first j jobs (see Eq. (1)). We expand the pair $(j;\ell_j^1)$ to a quadruple $(j;\ell_j^1,\ell_j^2;Z(j;\ell_j^1))$ in the following development of the FPTAS. We remark that in the quadruple $(j;\ell_j^1,\ell_j^2;Z(j;\ell_j^1))$, the real variables are j and ℓ_j^1, while ℓ_j^2 and $Z(j;\ell_j^1)$ are "functions" in j and ℓ_j^1.

Step 1 Set

$$\delta \triangleq (1+\epsilon/2n). \tag{15}$$

A partial schedule is now described as the quadruple $(j;\ell_j^1,\ell_j^2;Z(j;\ell_j^1))$, where $a \le j \le n$, $\ell_j^1 = 0$ or $p_{\min} \le \ell_j^1 \le T_1$, $\ell_j^2 = T_2 + P_j - \ell_j^1$, and $Z(j;\ell_j^1)$ is the total weighted completion time of the first j jobs in the partial schedule associated with $(j;\ell_j^1)$. Therefore, $T_2 + p_a \le \ell_j^2 \le T_2 + P$ and $Z(\mathcal{J}_1) + w_a(T_2+p_a) \le Z(j;\ell_j^1) \le W$, where W denotes the total weighted completion time of all the n jobs in any feasible schedule.[4] Let L^1, U^1 (L^2, U^2; L^3, U^3, respectively) denote the above lower and upper bounds on ℓ_j^1 (ℓ_j^2; $Z(j;\ell_j^1)$, respectively); let

$$r_i \triangleq \lfloor \log_\delta (U^i/L^i) \rfloor, \text{ for } i = 1,2,3,$$

and split the interval $[L^i, U^i]$ into subintervals

$$I_1^i = [L^i, L^i\delta],\ I_2^i = (L^i\delta, L^i\delta^2],\ \ldots,\ I_{r_i}^i = (L^i\delta^{r_i}, U^i], \text{ for } i = 1,2,3.$$

We define a three-dimensional box

$$B_{i_1,i_2,i_3} \triangleq I_{i_1}^1 \times I_{i_2}^2 \times I_{i_3}^3, \text{ for } (i_1,i_2,i_3) \in (\{0\}\cup[r_1]) \times [r_2] \times [r_3], \tag{16}$$

where $I_0^1 \triangleq [0,0]$ and $[r] \triangleq \{1,2,\ldots,r\}$ for any positive integer r. Each j such that $a \le j \le j_3$ or $j = n$ is associated with a box B_{i_1,i_2,i_3}, denoted as j-B_{i_1,i_2,i_3} for simplicity, where $(i_1,i_2,i_3) \in (\{0\}\cup[r_1]) \times [r_2] \times [r_3]$; all these boxes are initialized empty.

Step 2 The starting partial schedule is described as $(a;\ell_a^1,\ell_a^2;Z(a;\ell_a^1))$ in Eq. (11), which is then saved in the box a-B_{i_1,i_2,i_3} where $\ell_a^1 \in I_{i_1}^1$, $\ell_a^2 \in I_{i_2}^2$, and $Z(a;\ell_a^1) \in I_{i_3}^3$.

In general, for each non-empty box j-B_{i_1,i_2,i_3} with $a \le j < j_3$, denote the saved quadruple as $(j;\ell_j^1,\ell_j^2;Z(j;\ell_j^1))$, which describes a partial schedule on the first j jobs such that there is no machine idling period in the earlier schedule, ℓ_j^1 is the maximum job completion time in the earlier schedule, ℓ_j^2 is the maximum job completion time in the later schedule, and $Z(j;\ell_j^1)$ is the total weighted completion time of the jobs in the partial schedule. The algorithm Approx(a,ϵ) performs the same as the exact algorithm DP-2 to assign the next job J_{j+1} of $\mathcal{J}_2 \cup \mathcal{J}_3$ to generate at most three new partial schedules each described as a quadruple $(j+1;\ell_{j+1}^1,\ell_{j+1}^2;Z(j+1;\ell_{j+1}^1))$. Furthermore, if a non-empty machine idling period is inserted in the earlier schedule of a new partial schedule, then it is directly completed optimally into a full schedule using Lemma 2.2(d). For each resultant $(j+1;\ell_{j+1}^1,\ell_{j+1}^2;Z(j+1;\ell_{j+1}^1))$ (the same for $(n;\ell_n^1,\ell_n^2;Z(n;\ell_n^1))$) the algorithm checks whether or not the box $(j+1)$-B_{i_1,i_2,i_3} is empty, where $\ell_{j+1}^1 \in I_{i_1}^1$, $\ell_{j+1}^2 \in I_{i_2}^2$, and $Z(j+1;\ell_{j+1}^1) \in I_{i_3}^3$; if it is empty, then the quadruple is saved in the box, otherwise the box is updated to save the quadruple having a smaller ℓ_{j+1}^1 between the old and the new ones.

Step 3 For each box n-B_{i_1,i_2,i_3}, where $(i_1,i_2,i_3) \in (\{0\}\cup[r_1]) \times [r_2] \times [r_3]$, if there is a saved quadruple $(n;\ell_n^1,\ell_n^2;Z(n;\ell_n^1))$, then there is a constrained full schedule with the total weighted completion time $Z(n;\ell_n^1)$. Here the constraint is that the job J_a starts processing at time T_2. The algorithm Approx(a,ϵ) scans through all the boxes associated with n, and all those full schedules completed directly from a saved quadruple, and returns the quadruple having the smallest total weighted completion time, denoted as Z_n^a. The corresponding constrained full schedule can be backtracked.

Theorem 4.2 *The algorithm* Approx(a,ϵ) *is an* $O\left(\frac{n^4}{\epsilon^3}\log T_1 \log P \log W\right)$*-time* $(1+\epsilon)$*-approximation for the rescheduling problem* $(1, h_1 \mid \Delta_{\max} \le k \mid \sum_{j=1}^n w_j C_j)$*, under the constraint that the job* J_a *starts processing at time* T_2*, where* n *is the number of jobs, the machine is unavailable in* $[T_1,T_2]$*,* P *is the total job processing time, and* W *is the total weighted completion time of any feasible schedule.*

Proof The proof of the performance ratio is done by induction. Assume that $(a;\ell_a^{1*},\ell_a^{2*};Z(a;\ell_a^{1*})) \to (a+1;\ell_{a+1}^{1*},\ell_{a+1}^{2*};Z(a+1;\ell_{a+1}^{1*})) \to \cdots \to (j_3;\ell_{j_3}^{1*},\ell_{j_3}^{2*};Z(j_3;\ell_{j_3}^{1*})) \to (n;\ell_n^{1*},\ell_n^{2*};Z(n;\ell_n^{1*}))$ is the path of quadruples computed by the exact algorithm DP- 2 in Theorem 4.1 that leads to the constrained optimal solution for the problem $(1, h_1 \mid \Delta_{\max} \le k \mid \sum_{j=1}^n w_j C_j)$. The induction statement is for each j, $a \le j \le j_3$ or $j = n$, there is a quadruple $(j;\ell_j^1,\ell_j^2;Z(j;\ell_j^1))$ saved by the algorithm Approx(a,ϵ), such that $\ell_j^1 \le \ell_j^{1*}$, $\ell_j^2 \le \ell_j^{2*}\delta^j$, $Z(j;\ell_j^1) \le Z(j;\ell_j^{1*})\delta^j$.

The base case is $j = a$, and the statement holds since there is only one partial schedule on the first a jobs, described as $(a;\ell_a^1,\ell_a^2;Z(a;\ell_a^1))$ in Eq. (11). We assume the induction statement holds for j, where $a \le j \le j_3$, that is, there is a quadruple $(j;\ell_j^1,\ell_j^2;Z(j;\ell_j^1))$ saved by the algorithm Approx(a,ϵ), such that

$$\ell_j^1 \le \ell_j^{1*},\ \ell_j^2 \le \ell_j^{2*}\delta^j, \text{ and } Z(j;\ell_j^1) \le Z(j;\ell_j^{1*})\delta^j. \tag{17}$$

[4] In fact, W can be any upper bound on the optimal total weighted completion time of all the n jobs.

When $j < j_3$, from this particular quadruple $(j; \ell_j^1, \ell_j^2; Z(j; \ell_j^1))$, we continue to assign the job $J_{j+1} \in \mathcal{J}_2 \cup \mathcal{J}_3$ as in the exact algorithm DP- 2 in Sect. 4.1. In Case 1 where J_{j+1} is added in the later schedule to obtain a partial schedule described as in Eq. (12), we have

$$\ell_{j+1}^1 = \ell_j^1, \ell_{j+1}^2 = \ell_j^2 + p_{j+1}, \text{ and } Z(j+1; \ell_{j+1}^1) = Z(j; \ell_j^1) + w_{j+1}(\ell_j^2 + p_{j+1})). \quad (18)$$

Assume this quadruple falls in the box $(j+1)$-B_{i_1,i_2,i_3}, then the triple $(j+1; \hat{\ell}_{j+1}^1, \hat{\ell}_{j+1}^2; \hat{Z}(j+1; \ell_{j+1}^1))$ saved in this box by the algorithm Approx(a, ϵ) must have

$$\hat{\ell}_{j+1}^1 \le \ell_{j+1}^1, \hat{\ell}_{j+1}^2 \le \ell_{j+1}^2 \delta, \text{ and } \hat{Z}(j+1; \ell_{j+1}^1) \le Z(j+1; \ell_{j+1}^1)\delta. \quad (19)$$

If $(j; \ell_j^{1*}, \ell_j^{2*}; Z(j; \ell_j^{1*}))$ leads to $(j+1; \ell_{j+1}^{1*}, \ell_{j+1}^{2*}; Z(j+1; \ell_{j+1}^{1*}))$ also by adding J_{j+1} in the later schedule, then we have

$$\ell_{j+1}^{1*} = \ell_j^{1*}, \ell_{j+1}^{2*} = \ell_j^{2*} + p_{j+1}, \text{ and } Z(j+1; \ell_{j+1}^{1*}) = Z(j; \ell_j^{1*}) + w_{j+1}(\ell_j^{2*} + p_{j+1})). \quad (20)$$

From Eqs. (17–20) and $\delta > 1$, we have

$$\hat{\ell}_{j+1}^1 \le \ell_{j+1}^{1*}, \hat{\ell}_{j+1}^2 \le \ell_{j+1}^{2*} \delta^{j+1}, \text{ and } \hat{Z}(j+1; \ell_{j+1}^1) \le Z(j+1; \ell_{j+1}^{1*})\delta^{j+1}. \quad (21)$$

Similarly, in Case 2 where J_{j+1} is added in the earlier schedule without inserting a machine idling period to obtain a feasible partial schedule described as in Eq. (13), we can show that if $(j; \ell_j^{1*}, \ell_j^{2*}; Z(j; \ell_j^{1*}))$ leads to $(j+1; \ell_{j+1}^{1*}, \ell_{j+1}^{2*}; Z(j+1; \ell_{j+1}^{1*}))$ also in this way, then there is a saved quadruple $(j+1; \hat{\ell}_{j+1}^1, \hat{\ell}_{j+1}^2; \hat{Z}(j+1; \ell_{j+1}^1))$ by the algorithm Approx(a, ϵ) such that Eq. (21) holds.

In Case 3 where $\ell_j^1 + p_{j+1} < C_{j+1}(\pi^*) - k \le T_1$, J_{j+1} is added in the earlier schedule, and a non-empty machine idling period of length $C_{j+1}(\pi^*) - (\ell_j^1 + p_{j+1}) - k$ is inserted right before it, the algorithm continuously processes succeeding jobs in the earlier schedule as long as they fit (but up to J_{j_3}), and lastly process all the remaining jobs in the later schedule. When $j = j_3$, the algorithm completes the quadruple by assigning all the jobs of $\mathcal{J}_4$ in the later schedule, starting at time ℓ_j^2, to obtain a full schedule described as in Eq. (14). We can similarly show that if $(j; \ell_j^{1*}, \ell_j^{2*}; Z(j; \ell_j^{1*}))$ leads to $(n; \ell_n^{1*}, \ell_n^{2*}; Z(n; \ell_n^{1*}))$ directly, or if $(j; \ell_j^{1*}, \ell_j^{2*}; Z(j; \ell_j^{1*}))$ leads to $(j+1; \ell_{j+1}^{1*}, \ell_{j+1}^{2*}; Z(j+1; \ell_{j+1}^{1*}))$, and so on, then eventually to $(n; \ell_n^{1*}, \ell_n^{2*}; Z(n; \ell_n^{1*}))$, then there is also a saved quadruple $(n; \hat{\ell}_n^1, \hat{\ell}_n^2; \hat{Z}(n; \ell_n^1))$ by the algorithm Approx(a, ϵ) such that Eq. (21) holds with n replacing $j+1$. We therefore finish the proof of the induction statement.

Lastly we use the inequality $(1 + \epsilon/2n)^n \le 1 + \epsilon$ for $0 < \epsilon < 1$ to bound the ratio δ^n. That is, the total weighted completion time of the output full schedule by the algorithm Approx(a, ϵ) is

$$Z_n^a \le Z(n; \ell_n^{1*})\delta^n \le (1+\epsilon)Z(n; \ell_n^{1*}).$$

For the time complexity, note that for each j such that $a \le j \le j_3$ or $j = n$, there are $O(r_1 r_2 r_3)$ boxes associated with it; for a saved quadruple $(j; \ell_j^1, \ell_j^2; Z(j; \ell_j^1))$, it takes $O(1)$ time to assign the job J_{j+1}, leading to at most three new quadruples $(j+1; \ell_{j+1}^1, \ell_{j+1}^2; Z(j+1; \ell_{j+1}^1))$, each is used to update the corresponding box associated with $j+1$. It follows that the total running time is $O(n r_1 r_2 r_3)$. Note that from the definitions of r_i's, and $\log \delta = \log(1 + \epsilon/2n) \ge \epsilon/4n$, $O(n r_1 r_2 r_3) \subseteq O(n^4 \log T_1 \log P \log W/\epsilon^3)$. □

Enumerating all possible $O(n)$ choices of J_a, that is calling the algorithm Approx(a, ϵ) $O(n)$ times, the full schedule with the minimum Z_n^a is returned as the final solution. The overall algorithm is denoted as Approx(ϵ).

Corollary 3 *The algorithm* Approx(ϵ) *is an* $O\left(\frac{n^5}{\epsilon^3} \log T_1 \log P \log W\right)$*-time* $(1+\epsilon)$*-approximation for the problem* $(1, h_1 \mid \Delta_{\max} \le k \mid \sum_{j=1}^n w_j C_j)$.

4.3 An FPTAS for the general case

In this subsection, we derive an FPTAS for the general rescheduling problem $(1, h_1 \mid \Delta_{\max} \le k \mid \mu\Delta_{\max} + \sum_{j=1}^n w_j C_j)$ by invoking polynomial times the approximation algorithm Approx(a, ϵ) for the special case $\mu = 0$.

Recall that the job J_a starts processing at time T_2 by the algorithm Approx(a, ϵ). It follows that the maximum time deviation $\Delta_{\max}$ is bounded as $\Delta_a \le \Delta_{\max} \le k$, and we use $(1+\epsilon)$ to split the interval $[\Delta_a, k]$ into subintervals

$$I_1^4 = [\Delta_a, \Delta_a(1+\epsilon)], \; I_2^4 = (\Delta_a(1+\epsilon), \Delta_a(1+\epsilon)^2], \ldots, \; I_{r_4}^4 = (\Delta_a(1+\epsilon)^{r_4}, k],$$

where $r_4 \triangleq \lfloor \log_{1+\epsilon}(k/\Delta_a) \rfloor$.

Let $k_i = \Delta_a(1+\epsilon)^i$, for $i = 1, 2, \ldots, r_4$, and $k_{r_4+1} = k$. Using k_i as the upper bound on the maximum time deviation, that is, k_i replaces k, our algorithm calls Approx(a, ϵ) to solve the problem $(1, h_1 \mid \Delta_{\max} \le k_i \mid \sum_{j=1}^n w_j C_j)$, by returning a constrained full schedule of the total weighted completion time within $(1+\epsilon)$ of the minimum. We denote this full schedule as $\sigma(a, k_i)$. The final output full schedule is the one of the minimum objective function value among $\{\sigma(a, k_i) \mid 1 \le a \le j_1, \; 1 \le i \le r_4 + 1\}$, which is denoted as σ^ϵ with the objective function value Z^ϵ. We denote our algorithm as $\text{APPROX}^\mu(\epsilon)$.

Theorem 4.3 *The algorithm* $\text{APPROX}^\mu(\epsilon)$ *is an* $O\left(\frac{n^5}{\epsilon^4}\log T_1 \log P \log W \log k\right)$*-time* $(1+\epsilon)$*-approximation for the general rescheduling problem* $(1, h_1 \mid \Delta_{\max} \le k \mid \mu\Delta_{\max} + \sum_{j=1}^n w_j C_j)$*, where n is the number of jobs, the machine is unavailable in* $[T_1, T_2]$*, P is the total job processing time, and W is the total weighted completion time of any feasible schedule.*

Proof Let σ^* denote an optimal schedule for the problem with the objective function value Z^*, satisfying Lemmas 2.1 and 2.2. Suppose the job starts processing at time T_2 in σ^* is J_a, and the maximum time deviation in σ^* is $\Delta^*_{\max} = k^* \le k$. We thus conclude that σ^* is also a constrained optimal schedule for the problem by replacing the upper bound k with k^*, i.e., $(1, h_1 \mid \Delta_{\max} \le k^* \mid \mu\Delta_{\max} + \sum_{j=1}^n w_j C_j)$, under the constraint that J_a starts processing at time T_2.

Consider the value $k_i = \Delta_a(1+\epsilon)^i$ such that $\Delta_a(1+\epsilon)^{i-1} \le k^* \le \Delta_a(1+\epsilon)^i$. Clearly, for the schedule $\sigma(a, k_i)$ found by the algorithm $\text{APPROX}(a, \epsilon)$ to the problem $(1, h_1 \mid \Delta_{\max} \le k_i \mid \sum_{j=1}^n w_j C_j)$, it has the total weighted completion time $Z(\sigma) \le (1+\epsilon)Z(\sigma^*)$ and the maximum time deviation $\Delta_{\max}(\pi^*, \sigma) \le k_i \le k^*(1+\epsilon)$. It follows that

$$
\begin{aligned}
Z^\epsilon &\le \mu\Delta_{\max}(\pi^*, \sigma) + Z(\sigma) \\
&\le \mu k^*(1+\epsilon) + (1+\epsilon)Z(\sigma^*) = (1+\epsilon)Z^*.
\end{aligned}
$$

Note that we call the algorithm $\text{APPROX}(a, \epsilon)$ for all possible values of a, and for each a we call the algorithm $r_4 + 1 = O(\frac{1}{\epsilon}\log k)$ times [using the inequality $\log(1+\epsilon) \ge \frac{1}{2}\epsilon$]. Thus from Theorem 4.2 the total running time of the algorithm $\text{APPROX}^\mu(\epsilon)$ is in

$$
O\left(\frac{n^5}{\epsilon^4}\log T_1 \log P \log W \log k\right).
$$

This finishes the proof of the theorem. □

5 Concluding remarks

We investigated a rescheduling problem where a set of jobs has already been scheduled to minimize the total weighted completion time on a single machine, but a disruption causes the machine to become unavailable for a given time interval. The production planner needs to reschedule the jobs without excessively altering the originally planned schedule. The degree of alteration is measured as the maximum time deviation for all the jobs between the original and the new schedules. We studied a general model where the maximum time deviation is taken both as a constraint and as part of the objective function. We presented a pseudo-polynomial time exact algorithm based on dynamic programming and an FPTAS. We remark that the FPTAS calls polynomial (that is, $O(\log k/\epsilon)$) times another FPTAS for the special case where the maximum time deviation is not taken as part of the objective function; this special case gives us room to properly sample the maximum time deviation (that is, polynomial vs. exponential).

In the current rescheduling model, the machine unavailability is represented as a single time interval. It would be interesting to generalize our results to the case of multiple time intervals (Yin et al. 2016), for which we are not aware of any existing approximability results. Besides a single machine, one might want to investigate the other machine environments for which the original optimal schedule can be obtained in polynomial time, for example, the well-known two-machine flow-shop which also has numerous applications. In the literature, multiple parallel identical machine scheduling has been studied, for which (only) a near optimal schedule is used as the original schedule (Qi et al. 2006; Yin et al. 2016).

It would also be interesting to generalize our results to the case of stochastic machine unavailability, such as the rescheduling environment discussed in Yin et al. (2017) where the machine becomes unavailable starting at time T_1 and the unavailability lasts for a period of time with a certain probability. The goal of rescheduling is to minimize the sum of the expected weighted maximum time deviation and the expected total weighted completion time.

Acknowledgements All authors were supported by NSERC Canada. Additionally, W.L. was supported by K. C. Wong Magna Fund in Ningbo University, the China Scholarship Council (Grant No. 201408330402), and the Ningbo Natural Science Foundation (2016A610078); T.L. was supported by NSF China (Grant Nos. 71701162 and 71371129); G.L. was supported by NSF China (Grant Nos. 61471124 and 61672323).

A Proof of Lemma 2.1

Proof By contradiction, assume (J_i, J_j) is the first pair of jobs for which J_i precedes J_j in π^*, i.e., $p_i/w_i \le p_j/w_j$, but J_j immediately precedes J_i in the earlier schedule of σ^*. Let σ' denote the new schedule obtained from σ^* by swapping J_j and J_i. If $C_j(\sigma') \ge C_j(\pi^*)$, then $C_i(\sigma') \ge C_i(\pi^*)$ too, and thus $\Delta_j(\sigma', \pi^*) \le \Delta_i(\sigma', \pi^*) < \Delta_i(\sigma^*, \pi^*)$ due to $p_j > 0$; if $C_j(\sigma') < C_j(\pi^*)$ and $C_i(\sigma') \le C_i(\pi^*)$, then $\Delta_i(\sigma', \pi^*) \le \Delta_j(\sigma', \pi^*) < \Delta_j(\sigma^*, \pi^*)$ due to $p_i > 0$; lastly if $C_j(\sigma') < C_j(\pi^*)$ and $C_i(\sigma') > C_i(\pi^*)$, then $\Delta_i(\sigma', \pi^*) < \Delta_i(\sigma^*, \pi^*)$ and $\Delta_j(\sigma', \pi^*) < \Delta_j(\sigma^*, \pi^*)$. That is, σ' is also a feasible reschedule.

Furthermore, the weighted completion times contributed by J_i and J_j in σ' is no more than those in σ^*, implying the optimality of σ'. It follows that, if necessary, after a sequence of job swappings, we will obtain an optimal reschedule in which the jobs in the earlier schedule are in the same order as they appear in π^*. This proves the part (a).

For the second part (b) of the lemma, similarly by contradiction we assume (J_i, J_j) is the first pair of jobs for which J_i precedes J_j in π^*, i.e., $p_i/w_i \leq p_j/w_j$, but J_j immediately precedes J_i in the later schedule of σ^*. Let σ' denote the new schedule obtained from σ^* by swapping J_j and J_i. If $C_j(\sigma') \geq C_j(\pi^*)$, then $C_i(\sigma') \geq C_i(\pi^*)$ too, and thus $\Delta_j(\sigma', \pi^*) \leq \Delta_i(\sigma', \pi^*) < \Delta_i(\sigma^*, \pi^*)$ due to $p_j > 0$; if $C_j(\sigma') < C_j(\pi^*)$ and $C_i(\sigma') \leq C_i(\pi^*)$, then $\Delta_i(\sigma', \pi^*) \leq \Delta_j(\sigma', \pi^*) < \Delta_j(\sigma^*, \pi^*)$ due to $p_i > 0$; lastly if $C_j(\sigma') < C_j(\pi^*)$ and $C_i(\sigma') > C_i(\pi^*)$, then $\Delta_i(\sigma', \pi^*) < \Delta_i(\sigma^*, \pi^*)$ and $\Delta_j(\sigma', \pi^*) < \Delta_j(\sigma^*, \pi^*)$. That is, σ' is also a feasible reschedule.

Furthermore, the weighted completion times contributed by J_i and J_j in σ' is no more than those in σ^*, implying the optimality of σ'. If follows that, if necessary, after a sequence of job swappings we will obtain an optimal reschedule in which the jobs in the earlier schedule are in the same order as they appear in π^*. □

B Proof of Lemma 2.2

Proof Item (a) is a direct consequence of Lemma 2.1, since the jobs in the earlier schedule are in the same order as they appear in π^*, which is the WSPT order. That is, for the job J_j in the earlier schedule of σ^*, if all the jobs J_i, for $i = 1, 2, \ldots, j-1$, are also in the earlier schedule, then $C_j(\sigma^*) = C_j(\pi^*)$; otherwise, $C_j(\sigma^*) < C_j(\pi^*)$.

For item (b), assume the machine idles before processing the jobs J_{j_1} and J_{j_2}, with J_{j_1} preceding J_{j_2} in the earlier schedule. We conclude from Lemma 2.1 and item (a) that if $\Delta_{j_2} < \Delta_{j_1} \leq \Delta_{\max}$ (or $\Delta_{j_1} < \Delta_{j_2} \leq \Delta_{\max}$, respectively), then moving the starting time of the job J_{j_2} (J_{j_1}, respectively) one unit ahead will maintain the maximum time deviation and decrease the total weighted completion time, which contradicts the optimality of σ^*. It follows that we must have $\Delta_{j_1} = \Delta_{j_2}$; in this case, if there is any job J_j with $j_1 < j < j_2$ in the later schedule, it can be moved to the earlier schedule to decrease the total weighted completion time, which again contradicts the optimality of σ^*. Therefore, there is no job J_j with $j_1 < j < j_2$ in the later schedule, which together with $\Delta_{j_1} = \Delta_{j_2}$ imply that the machine does not idle before processing the job J_{j_2}.

Item (c) is clearly seen for the same reason used in the last paragraph, that firstly their time deviations have to be the same, and secondly if this time deviation is less than $\Delta_{\max}$, one can then move their starting time one unit ahead to decrease the total weighted completion time while maintaining the maximum time deviation, thus contradicting the optimality of σ^*.

Item (d) is implied by Item (c), given that all the job processing times are positive.

Let the job in the earlier schedule right after the idle time period be J_j. clearly, there is a job J_a with $S_a(\pi^*) < S_j(\sigma^*)$ in the later schedule of σ^*, due to the machine idling. The time deviation for J_a is $\Delta_a > T_2 - S_a(\pi^*)$. If $S_j(\pi^*) < T_2$, then $\Delta_{\max} = \Delta_j < T_2 - S_j(\sigma^*) < T_2 - S_a(\pi^*) < \Delta_a$, a contradiction. This proves item (e) that $S_j(\pi^*) \geq T_2$.

Item (f) is clearly seen from Lemma 2.1 and the optimality of σ^*, that if the machine idles then it can start processing the jobs after the idling period earlier to decrease the total weighted completion time, while maintaining or even decreasing the maximum time deviation.

From item (f) and the second part of Lemma 2.1, we see that the deviation times of the jobs in the later schedule are non-increasing. Therefore, item (g) is proved. □

References

Aytug, H., Lawley, M. A., McKay, K., Mohan, S., & Uzsoy, R. (2005). Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, *161*, 86–110.

Bean, J. C., Birge, J. R., Mittenthal, J., & Noon, C. E. (1991). Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research*, *39*, 470–483.

Clausen, J., Hansen, J., Larsen, J., & Larsen, A. (2001). Disruption management. *OR/MS Today*, *28*, 40–43.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annuals of Discrete Mathematics*, *5*, 287–326.

Hall, N. G., Liu, Z., & Potts, C. N. (2007). Rescheduling for multiple new orders. *INFORMS Journal on Computing*, *19*, 633–645.

Hall, N. G., & Potts, C. N. (2004). Rescheduling for job unavailability. *Operations Research*, *58*, 746–755.

Hall, N. G., & Potts, C. N. (2004). Rescheduling for new orders. *Operations Research*, *52*, 440–453.

Herroelen, W., & Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, *165*, 289–306.

Hoogeveen, H., Lenté, C., & T'kindt, V. (2012). Rescheduling for new orders on a single machine with setup times. *European Journal of Operational Research*, *223*, 40–46.

Lee, C.-Y. (1996). Machine scheduling with an availability constraint. *Journal of Global Optimization*, *9*, 395–416.

Liu, Z., & Lin, G. (2017). Personal communication.

Liu, Z., Lu, L., & Qi, X. (2017). Cost allocation in rescheduling with machine unavailable period. *European Journal of Operational Research*,. https://doi.org/10.1016/j.ejor.2017.09.015.

Liu, Z., & Ro, Y. K. (2014). Rescheduling for machine disruption to minimize makespan and maximum lateness. *Journal of Scheduling*, *17*, 339–352.

Luo, W., Luo, T., Goebel, R., Lin, G. (2017). On rescheduling due to machine disruption while to minimize the total weighted completion time. *CoRR*, abs/1701.07498

Qi, X., Bard, J. F., & Yu, G. (2006). Disruption management for machine scheduling: The case of SPT schedules. *International Journal of Production Economics*, *103*, 166–184.

Vieira, G. E., Herrmann, J. W., & Lin, E. (2003). Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *Journal of Scheduling*, *6*, 39–62.

Wang, D., Liu, F., & Jin, Y. (2017). A multi-objective evolutionary algorithm guided by directed search for dynamic scheduling. *Computers & Operations Research*, *79*, 279–290.

Wang, D., Liu, F., Wang, Y., & Jin, Y. (2015). A knowledge-based evolutionary proactive scheduling approach in the presence of machine breakdown and deterioration effect. *Knowledge Based Systems*, *90*, 70–80.

Yang, J., Qi, X., & Yu, G. (2005). Disruption management in production planning. *Naval Research Logistics*, *52*, 420–442.

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

Yin, Y., Wang, Y., Cheng, T. C. E., Liu, W., & Li, J. (2017). Parallel-machine scheduling of deteriorating jobs with potential machine disruptions. *Omega*, *69*, 17–28.

Yuan, J. J., & Mu, Y. (2007). Rescheduling with release dates to minimize makespan under a limit on the maximum sequence disruption. *European Journal of Operational Research*, *182*, 936–944.

Zhao, Q., & Yuan, J. J. (2013). Pareto optimization of rescheduling with release dates to minimize makespan and total sequence disruption. *Journal of Scheduling*, *16*, 253–260.

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