Cost Allocation in Rescheduling with Machine Unavailable Period

Cost Allocation in Rescheduling with Machine Unavailable Period

Zhixin Liu, Liang Lu, Xiangtong Qi

PII: S0377-2217(17)30805-6
DOI: 10.1016/j.ejor.2017.09.015
Reference: EOR 14693

To appear in: *European Journal of Operational Research*

Received date: 21 November 2015
Revised date: 1 May 2017
Accepted date: 12 September 2017

Please cite this article as: Zhixin Liu, Liang Lu, Xiangtong Qi, Cost Allocation in Rescheduling with Machine Unavailable Period, *European Journal of Operational Research* (2017), doi: 10.1016/j.ejor.2017.09.015

Highlights

- Optimization and approximation for rescheduling with machine unavailability.
- Model the rescheduling problem as a cooperative sequencing game.
- Direct construction of the Shapley value for the sequencing game.
- Recognize relationship between core allocation and Shapley value.
- Computational study for the optimization problem and Shapley value.

Cost Allocation in Rescheduling with Machine Unavailable Period

Zhixin Liu
Corresponding Author
Department of Management Studies, College of Business
University of Michigan–Dearborn
19000 Hubbard Drive, Dearborn, Michigan 48126-2638, United States
E-mail: zhixin@umich.edu

Liang Lu
Logistics Research Centre, School of Management & Languagues
Heriot-Watt University
Edinburgh, EH144AS, United Kingdom
E-mail: nicowish@gmail.com

Xiangtong Qi
Department of Industrial Engineering and Logistics Management
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
E-mail: ieemqi@ust.hk

Submitted: November 21, 2015
Revised: January 4, 2017
Revised: September 16, 2017

Abstract

We study a rescheduling problem faced by multiple jobs owners sharing a single machine, where jobs need to be rescheduled, when the machine becomes unavailable for a period of time. The disruption caused by any new schedule is restricted such that the difference between the completion times in the initial and the new schedules of any job is no more than a given threshold. A natural way to reschedule is to process the jobs in the initial sequence, each as early as possible. This defines a feasible schedule over which cost saving can potentially be achieved by optimal rescheduling as long as the cost saving can be fairly shared by job owners. We define a cooperative game for job owners accordingly, to share the cost saving. Given that the optimization problem is computationally intractable, we find several optimal properties and develop an optimal pseudopolynomial time dynamic programming algorithm for rescheduling. We provide a simple closed form core allocation of the total cost saving for all the jobs, and also provide the Shapley value of the game in a computable form. Then we computationally evaluate the extra cost caused by machine unavailability, the cost saving from optimization relative to the naturally constructed schedule, the likelihood for the Shapley value to be a core allocation, and how the Shapley value allocates cost saving among job owners. Managerial insights are derived from the computational studies. This work contributes to the literature by explicitly incorporating two classic scheduling topics: sequencing game and rescheduling.

Key words: scheduling; rescheduling; machine unavailability; sequencing game; core.

1 Introduction

In the framework of cooperative game theory, researchers have studied various models for the possibility of collaboration in sequencing and scheduling problems. All such works share the following basic setting. There are multiple players each owning one or multiple jobs to be processed on a common set of machines. There is a given schedule in which some or all players can swap their jobs to obtain a new schedule with cost reduction. One important problem to address is how to allocate the achieved cost saving in a mutually beneficial and fair way among players. The focus of the above mentioned works is on designing a reasonable allocation scheme, e.g., a core allocation, to enable the collaboration of all the participating players who act from self-interest. If such an allocation exists, then cooperation is incentivized and centralized optimum can be achieved by a coordinated decision.

One particular case arises when there is a need of modifying the given schedule caused by external disruptions such as an unplanned machine breakdown. When a preplanned schedule has to be changed due to a disruption, the disruption also triggers an opportunity for rescheduling to reduce cost by collaboration, as long as it maintains the degree of change within a certain range. In fact, ensuring stability in rescheduling has been long recognized as an important requirement in centralized optimization. The purpose of this paper is to develop a model to demonstrate the possible benefit of collaboration in rescheduling after a disruption, under the condition that the new schedule is not too much deviated from the original schedule.

Specifically, we consider a typical rescheduling problem caused by unexpected machine unavailability. There are a set of jobs and a single machine where the jobs have been scheduled to be processed consecutively on the machine, with the total weighted completion time minimized. At the execution stage the machine becomes unavailable for a time interval, calling for rescheduling the jobs. In the new schedule, the aim is to optimize the original objective, minimizing total weighted completion time, under the condition that each job has a new completion time with a limited deviation from the original one. Note that we assume each job is owned by a different job owner. A natural way of rescheduling is to process the jobs in the initial sequence, each as early as possible, which defines a feasible schedule over which cost saving can potentially be achieved by optimal rescheduling. We define a cooperative game for the rescheduling problem accordingly.

We achieve the following technical contributions. First, for the problem with a centralized decision maker, which is a new scheduling problem on its own and is weakly *NP*-hard (Garey and Johnson 1979), we identify several optimal properties and develop an optimal pseudopolynomial time dynamic programming algorithm. Second, for the newly defined rescheduling game, we show that the game is with nonempty core by designing a simple closed form core allocation, and provide the Shapley value (Shapley 1953) of the game in a computable form. Third, we computationally evaluate the extra cost caused by machine unavailability, the cost saving from optimization relative to a naturally constructed schedule, the likelihood for the Shapley value to be a core allocation, and how the Shapley value allocates cost saving among job owners. Managerial insights are derived from extensive computational studies.

The rest of this paper is organized as follows. Relevant literature is reviewed in Section 2. In Section 3 we formally define the problem. Several optimal properties are recognized and an optimal dynamic programming algorithm is developed for the rescheduling problem in Section 4. Then in Section 5 we provide a core allocation and find the Shapley value for the defined rescheduling game. Computational studies are carried in Section 6. Finally, we conclude in Section 7.

2 Literature Review

We preceded to review relevant research in three streams: supply chain scheduling, scheduling problems that use cooperative game theory concepts to study coordination/cooperation issues, and rescheduling.

This work exploits scheduling issues with game theoretical interactions among involved parties, and thus belongs to the stream of supply chain scheduling, which quantifies benefits achieved through centralized decisions over scheduling issues in multiple tiers of a supply chain. We refer readers to a recent review by Aydinliyim and Vairaktarakis (2011). Supply chain scheduling is first studied by Hall and Potts (2003), who examine the coordinated scheduling, batching, and delivery decisions in a supply chain with a supplier and multiple manufacturers. Agnetis et al. (2006) investigate coordinated sequencing decisions for a supplier and multiple manufacturers who each has a preferred schedule determined by its own costs and

constraints. Dawande et al. (2006) study how a supplier and a distributor in a supply chain can coordinate their conflicted schedules. Manoj et al. (2008) exploit the benefits of coordinated decisions in a supply chain with a manufacturer and a distributor, where the distributor incurs inventory holding cost and sells to multiple retailers. Selvarajah and Steiner (2008) provide approximation algorithms for batch scheduling decisions of a supplier who sells multiple products and delivers them in batches. Steiner and Zhang (2009) develop approximation algorithms to minimize the total weighted number of late jobs in batch scheduling decisions for a supply chain with a supplier and multiple customers. Manoj et al. (2012) examine the benefits of coordinated scheduling decisions in a two stage production system with between-stage buffers, where the first stage incurs cost of total completion time and the second stage incurs cost of tardiness and resequencing.

Next we review relevant works on scheduling problems that use cooperative game theory concepts to study coordination/cooperation issues; see the surveys of Curiel et al. (2002), Hall and Liu (2008), and Aydinliyim and Vairaktarakis (2011). Curiel et al. (1989) study a one machine sequencing game where an initial arbitrary sequence of jobs is given, and rescheduling of jobs is allowed to reduce the scheduling cost and the saving can be shared by job owners, where the scheduling cost of each job is linearly nondeceasing with its completion time. Curiel et al. (1994) generalize the linear cost sequencing game to the case with a general nondecreasing scheduling cost. Slikker (2006) studies a relaxed version of the one machine sequencing game in that rescheduling of jobs in any coalition is allowed as long as it does not delay the processing of any job outside the coalition. Hall and Liu (2010) investigate how multiple distributors can coordinate their decisions when they order capacity from the manufacturer who makes both capacity allocation and scheduling decisions. Aydinliyim and Vairaktarakis (2010) study the coordination of manufacturers when they outsource scheduling capacity from a common third party to minimize the total of weighted completion time and booking cost. Cai and Vairaktarakis (2012) examine how to coordinate the capacity allocation and scheduling decisions and allocate the resulted savings for multiple manufacturers and a capacity owner, in the presence of booking, overtime, and tardiness cost. Aydinliyim and Vairaktarakis (2013) consider cooperation mechanisms of multiple agents who schedule their workloads over two resources: the in-house one is agent-specific, and the other one is more

flexible. Aydinliyim et al. (2014) investigate how multiple manufacturers can cooperate their capacity booking decisions towards a common capacity owner to minimize their holding, tardiness, and booking costs.

The area of rescheduling is rich, and our review focuses on rescheduling with machine disruption/unavailability. Scheduling with machine breakdown or unavailability has been an active research field for more than three decades; e.g., see the earliest works of Adiri et al. (1989) and Lee and Liman (1992), and the subsequent works of Lee (1996), Kacem and Chu (2008), and Huo et al. (2014). The starting focus of the major work has been on designing optimal algorithms for a certain classic scheduling criterion. Later on, in the context of rescheduling after a disruption, which can go beyond machine breakdown, ensuring the stability or minimal deviation from an initially given schedule becomes an additional issue to measure a new schedule, e.g., Bean et al. (1991), Unal et al. (1997), Aytug et al. (2004), Hall and Potts (2004, 2010), Azizoğlu and Alagöz (2005), Qi et al. (2006), Hall et al. (2007), Yang (2007), Yuan and Mu (2007), and Liu and Ro (2014). In all the cases, however, the authors assume a single decision maker with a full control of all the jobs. To the best of our knowledge, our paper is the first that studies the game-theoretic models for rescheduling problems after a disruption. For application, the rescheduling problem we consider falls into the field of disruption management (Clausen et al. 2001, Yu and Qi 2004), which addresses how to effectively revise a given operational schedule under various disruptions in the execution stage. Disruption management has applications in many areas such as airlines (Yu et al. 2003), healthcare (Thompson et al. 2009), and liner shipping (Li et al. 2015).

3 Definition

Consider a set of jobs, denoted $J = \{1, \ldots, n\}$, to be processed on a single machine without preemption. These jobs belong to different job owners. We assume that each owner owns a single job. In the specific game model considered, for the core allocation, the payoff is based on the contribution of and is additive over each individual job. If an owner owns multiple jobs, that owner can be equivalently treated as multiple single-job owners. Each owner or equivalently its job is referred to as a player. Each player is interested in minimizing the scheduling cost of its own job. A set of players is called a coalition if they cooperate in

scheduling their jobs. The coalition containing all the players is called the grand coalition, denoted $\mathcal{N}$.

Let p_j denote the processing time of job j, and w_j its weight as a priority factor for the job to be processed earlier, for $j = 1, \ldots, n$. We assume that all values of p_j and w_j are known integers. This restriction is necessary for us to develop a pseudopolynomial time optimal algorithm for the rescheduling problem, though it is a common assumption in the literature, for the perspective of computing. Further, we let $p_{\min} = \min_{j \in J}\{p_j\}$ and $P = \sum_{j \in J} p_j$. We consider the total weighted completion time as the scheduling cost. Such a cost can find its practical meaning in both manufacturing and service industries (Pinedo 2012). For example, the cost indicates the work-in-process inventory level, where the weight may model the material cost of each job, and thus a minimized total weighted completion time reduces inventory cost. Also, the cost can represent service level, where the job completion time is related to customer waiting times and the weight may represent a customer's relative importance to the service provider.

Following the convention of rescheduling literature, we assume that the jobs in J have been previously scheduled to minimize their total weighted completion time. Let T_1 and T_2 ($T_2 \geq T_1$) denote the start and end of a time period, during which the machine is unavailable for processing jobs. We assume that at time zero we know T_1 and T_2, prior to processing, but after making the initial schedule of the jobs of J. In fact, if after time zero T_1 and T_2 are both known, then we can remove the jobs of J having already been processed, complete any partially processed job in J, and update J, n, T_1 and T_2 accordingly. Note that we assume preemption is not allowed.

Let π^* denote an initial optimal schedule, and y^* its total scheduling cost. Assuming an initial optimal schedule before disruption is relevant to the motivation of the study, i.e., disruption recovery. Usually the interest of disruption recovery is to understand the impact of a disruption on a given initial optimal schedule. Technically, assuming an optimal initial schedule defines an optimization problem reasonably solvable. On the other hand, from a game perspective, whether the initial sequence is optimal or not does not matter: the core allocations and Shapley value we develop in this work do not depend on any specific initial schedule and optimization algorithm for rescheduling.

For each job j in a feasible schedule σ of the jobs of J after rescheduling for the machine unavailability, we define $S_j(\sigma)$ the start time of job j, $C_j(\sigma) = S_j(\sigma)+p_j$ the completion time of job j, and $\Delta_j(\pi^*, \sigma) = |C_j(\sigma) - C_j(\pi^*)|$ the *time disruption* of job j. Let the *maximum time disruption* be $\Delta_{\max}(\pi^*, \sigma) = \max_{j\in J}\{\Delta_j(\pi^*, \sigma)\}$. When the meaning is clear from 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 disruption models any penalties that are associated with the changing of start and completion times of jobs. Finally, let σ^* denote an optimal schedule after the rescheduling for the grand coalition, and z^* its scheduling cost.

Following the $\alpha|\beta|\gamma$ classification scheme established by Graham et al. (1979), our rescheduling problem is denoted $1|a, \Delta_{\max} \leq k|\sum w_jC_j$, where $\alpha = 1$ represents a single machine; in the β field, a means that there is an availability constraint of the machine, and $\Delta_{\max} \leq k$ means that the maximum time disruption of job processing can be no more than k, where k is a prescribed integer that defines the maximum allowable time disruption for any schedule; and lastly $\gamma = \sum w_jC_j$ represents the total weighted completion time scheduling cost. Our use of a in the β field is consistent with Lee (1996).

Problem $1|a, \Delta_{\max} \leq k|\sum w_jC_j$ can be related to the rescheduling model for new orders introduced in Hall and Potts (2004). In fact, the unavailable interval $[T_1, T_2]$ can be regarded as a new job with a release date T_1, a processing time $p = T_2 - T_1$ and a sufficiently large weight $w \gg 0$. Then the rescheduling with machine unavailable period is a special version of rescheduling with just one new order with a release date. The later problem, denoted $1|a^*, \Delta_{\max} \leq k|\sum w_jC_j$, has an interesting practical meaning: it models the case where the disruption, such as caused by urgent maintenance, can be postponed at certain cost. We leave problem $1|a^*, \Delta_{\max} \leq k|\sum w_jC_j$ for future investigation.

One question for coordinated rescheduling is whether job owners indeed have the oversight and flexibility to engage in coalition formation with each other for rescheduling purposes. In many cases, such coordination needs to be facilitated by the owner of the capacity. In this sense, it is meaningful to include the capacity owner, i.e., the machine, as a player. However, the capacity owner's value is hard to measure using the cost of each job. Also, job owners may request or expect certain compensation from capacity owner for their delay, which is not considered by our model since the compensation may not be related to the cost

of each job. Considering the case where flights are delayed by unexpected events, airline companies reschedule passengers to meet their individual demand, under certain constraints such that not delaying other passengers. Here airline companies have an implicit objective: not losing customer goodwill. Also, airline companies may provide fixed condensation to delayed passengers. The specific gaming issues faced by each player and coalition for the rescheduling problem are introduced in Section 5.

4 Optimization and Approximation

In this section we first describe certain optimal properties that are helpful in finding an optimal schedule, and then develop an optimal dynamic programming algorithm. Note that the optimal properties we identify are not necessary, which means that there can be other optimal schedules not satisfying these properties.

Consider the initial optimal schedule π^* before the machine unavailability. For the scheduling cost $\sum w_j C_j$, schedule π^* is defined by a shortest weighted processing time first (SWPT) indexing rule, where jobs are sequenced by nondecreasing ratio of processing time to weight, p_j/w_j, each as early as possible (Smith 1956). For convenience, we assume that the jobs of J are indexed such that $p_1/w_1 \leq p_2/w_2 \leq \cdots \leq p_n/w_n$, and denote $\pi^* = (1, \ldots, n)$ accordingly.

Let $j_1 = \min\{j|C_j(\pi^*) > T_1\}$ be the first job in π^* completed after T_1, and $j_2 = \min\{j|S_j(\pi^*) > T_2\}$ be the first job in π^* started after T_2. In a feasible schedule σ after rescheduling, we denote the partial schedule of jobs completed no later than T_1 as σ_A, and the partial schedule of jobs started no earlier than T_2 as σ_B.

Next, we consider the values of T_1, T_2, and k that are meaningful and nontrivial to study. First, if $T_1 < p_{\min}$, then by moving job 1 to start at T_2 and processing order jobs consequently, an optimal scheduled is found. Second, if $T_1 \geq P$, then there is no need for rescheduling. Third, by the definition of j_1, jobs $1, \ldots, j_1$ in π^* cannot all be processed in σ_A. However, if $k < T_2 - S_{j_1}(\pi^*)$, then it is infeasible to schedule any of jobs $1, \ldots, j_1$ in σ_B. Therefore, we precede to assume that

$$p_{\min} \leq T_1 < P$$

and

$$k \geq T_2 - S_{j_1}(\pi^*).$$

An appealing and natural way to reschedule is to process jobs following the same SWPT sequence as in π^*, each as early as possible. It is not difficult to verify, such an obtained schedule would have the minimum of the maximum time disruption, among all feasible schedules, and hence is feasible. However, such a schedule may not be optimal. Intuitively, this is due to the fact that a period of machine idle time immediately preceding T_1 could be wasted without being used for job process, as can be seen in Example 1, where the heuristic schedule of the same SWPT sequence is denoted σ^H. This simple scheduling rule will be discussed further in Sections 5 and 6.

Example 1 $n = 3$; $p_1 = 3$, $p_2 = 7$, $p_3 = 4$; $w_1 = 4$, $w_2 = 9$, $w_3 = 5$; $T_1 = 6$, $T_2 = 7$; $k = 9$. Schedules π^*, σ^* and σ^H are depicted in Figure 1, where the processing time is specified in each task, and UN refers to the unavailable period of the machine. The total weighted completion times of schedules π^*, σ^* and σ^H are 172, 218 and 228, respectively.

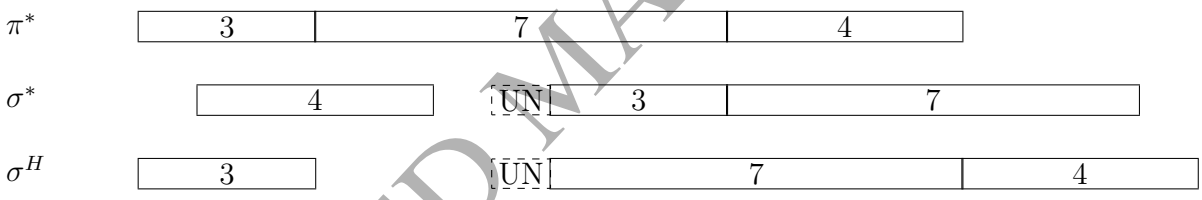

Figure 1: Gantt Chart for Example 1.

Liu and Ro (2014) study a rescheduling problem $1|a, \Delta_{\max} \leq k|L_{\max}$, where $L_{\max}$ refers to the maximum lateness scheduling cost. In single machine scheduling, for both costs $\sum w_j C_j$ and $L_{\max}$, an optimal schedule is defined by an indexing rule following which each job is processed as early as possible: SWPT sequence for $\sum w_j C_j$ and the earliest due date first (EDD) sequence for $L_{\max}$ (Jackson, 1955). Liu and Ro (2014) identify six optimal properties for problem $1|1, \Delta_{\max} \leq k|L_{\max}$, which also analogically hold for problem $1|a, \Delta_{\max} \leq k|\sum w_j C_j$. We provide the following Lemmas 1–6 with proofs in the appendices.

Lemma 1 *There exists an optimal schedule σ^* for problem $1|a, \Delta_{\max} \leq k|\sum w_j C_j$ in which (a) the jobs in σ_A^* are sequenced in the same order as in π^*; (b) the jobs in σ_B^* are sequenced in the same order as in π^*.*

An *inserted idle time period* is a period of machine idle time that immediately precedes the processing of a job. A job processed in σ_A of σ could be completed earlier than in π^*. In this case, due to the constraint on time disruption, the job might be immediately preceded by an inserted idle time period. Lemmas 2–4 are about inserted idle time periods in an optimal schedule of problem $1|a, \Delta_{\max} \leq k|\sum w_j C_j$.

Lemma 2 *There exists an optimal schedule σ^* for problem $1|a, \Delta_{\max} \leq k|\sum w_j C_j$, in which (a) the jobs in σ_A^* are processed with at most one inserted idle time period; (b) each job processed in σ_A^* after an inserted idle time period is processed exactly k time units earlier than in π^*; (c) the jobs processed in σ_A^* after an inserted idle time period are processed consecutively in π^*.*

Lemma 3 *There exists an optimal schedule σ^* for problem $1|a, \Delta_{\max} \leq k|\sum w_j C_j$ in which if a job is immediately preceded by an idle time period in σ_A^*, then in π^* the job has a start time later than T_2.*

Lemma 4 *There exists an optimal schedule σ^* for problem $1|a, \Delta_{\max} \leq k|\sum w_j C_j$, in which the jobs in σ_B^* are processed without any inserted idle time period.*

The inserted idle time period considered by Lemma 2, following its definition, should immediately precede a job processing. In σ_A of σ^*, there might exist another idle time period immediately preceding T_1. Example 1 shows how an inserted idle time period and another idle time period occur, and how the properties in Lemma 2–4 are satisfied by σ^*. Next, Lemma 5 is about the maximum time disruption of jobs in σ_B^B of σ^*.

Lemma 5 *There exists an optimal schedule σ^* for problem $1|a, \Delta_{\max} \leq k|\sum w_j C_j$, in which among all the jobs in σ_B^*, the first processed job has the maximum time disruption.*

Note that the proofs of Lemmas 1–5 actually indicate that there exists an optimal schedule satisfying the properties in Lemmas 1–5 simultaneously. Next we consider the minimum value of k that degenerates problem $1|a, \Delta_{\max} \leq k|\sum w_j C_j$ to problem $1|a, |\sum w_j C_j$.

Lemma 6 *In problem $1|a, \Delta_{\max} \leq k|\sum w_j C_j$, if $k \geq \max\{T_2, P - p_{\min}\}$, then an optimal schedule is found by solving problem $1|a, |\sum w_j C_j$.*

In view of Lemma 6, we assume $k \leq \max\{T_2, P - p_{\min}\}$.

Problem $1|a, \Delta_{\max} \leq k| \sum w_j C_j$ is at least binary *NP*-complete, following the complexity results of its special case $1|a, | \sum w_j C_j$ (Adiri et al. 1989). We next develop a pseudopolynomial time optimal algorithm, which indicates that the rescheduling problem is not unary *NP*-complete. We find an optimal schedule σ^* following the structure specified by Lemmas 1–5, as follows. In general, schedule σ^* contains three *blocks* of consecutively processed jobs in the same order as in π^*, with no idle time in each block. The first block starts from zero and completes earlier than T_1. The second block starts after the first block and completes no later than T_1. There is an inserted idle time period between the two blocks. Also, there could be machine idle time after the second block and before T_1. The time disruption of any job in the second block is exactly k. Finally, the third block starts at T_2.

Algorithm 1

Input

Given $p_1, \ldots, p_n$, $w_1, \ldots, w_n$, $\pi^* = (1, \ldots, n)$ in which jobs are indexed by an SWPT indexing rule, T_1, T_2 and k.

Initialization

Find $P = \sum_{j=1}^{n} p_j$, $j_1 = \min\{j|C_j(\pi^*) > T_1\}$, $j_2 = \min\{j|S_j(\pi^*) > T_2\}$, and $j_3 = \max\{j|C_j(\pi^*) - k \leq T_1\}$. Note that j_3 is the last processed job in π^* that can be scheduled in σ_A with a time disruption less than or equal to k.

Value Function

We denoted job i the first job in the second block preceded by an inserted idle time period. Note that in an optimal schedule, there may not exist inserted idle time, and we define $i = n+1$ to model this specific case.

$f_i(j, t) =$ the minimum total weighted completion time of any schedule of jobs $\{1, \ldots, j\} \cup \{i, \ldots, j_3\}$ in which

1. The maximum time disruption is no more than k.

2. Job i, if scheduled in σ_A, is processed with start time $S_i(\pi^*) - k$ and completion time $C_i(\pi^*) - k$, with an idle time period immediately preceding it. Feasibility requires that the time disruption of job i is no more than k when completed at time T_1, i.e.,

$C_i(\pi^*) - T_1 \leq k$. Also, according to Lemma 3, it is sufficient to consider $i \geq j_2$. The case without inserted idle time in σ_A is modelled by an artificial job $i = n+1$ that does not need to be scheduled.

3. Jobs $i+1, \ldots, j_3$, if scheduled, are processed immediately following job i in σ_A, without inserted idle. By Lemma 2, such scheduling of jobs $i, \ldots, j_3$ is sufficient to find an optimal schedule.

4. The maximum completion time of jobs processed before job i and in σ_A equals t, where $t \leq S_i(\pi^*) - k$. To model the case where there is no inserted idle time in σ_A, we define $S_{n+1}(\pi^*) = T_1 + k$.

Boundary Condition

$$f_i(0,t) = \begin{cases} \sum_{j=i}^{j_3} w_j(C_j(\pi^*) - k), & \text{if } 0 \leq t \leq S_i(\pi^*) - k, \\ \infty, & \text{otherwise,} \end{cases}$$

where $j_2 \leq i$ and $C_i(\pi^*) - k \leq T_1$, or $i = n+1$.

Optimal Solution Value

$\min_i\{f_i(n,t)\}$, where $j_2 \leq i$ and $C_i(\pi^*) - k \leq T_1$, or $i = n+1$.

Recurrence Relation

$$f_i(j,t) = \min \begin{cases} f_i(j-1,t) + w_j C_j, & \text{if } C_j - C_j(\pi^*) \leq k \text{ and } j \notin \{i, \ldots, j_3\}, \quad (1) \\ f_i(i-1,t), & \text{if } j = j_3 \text{ and } i \neq n+1, \quad (2) \\ f_i(j-1, t-p_j) + w_j t, & \text{if } t - p_j \geq 0,\ C_j(\pi^*) - t \leq k \text{ and } j < i, \quad (3) \\ \infty, & \text{otherwise,} \quad (4) \end{cases}$$

where $C_j = T_2 + \sum_{m=1}^{j} p_m - t - \sum_{m=i}^{\min\{j,j_3\}} p_m$ is the completion time of job j in σ_B.

Following Lemma 1, in the recurrence relation jobs are indexed by an SWPT indexing rule. Next we explain the four equations in the recurrence relation. First, equation (1) schedules job j in σ_B with completion time C_j and cost $w_j C_j$, given that its time disruption is no more than k, and that job j does not belong to $\{i, \ldots, j_3\}$. Following Lemma 4, no inserted idle time period is considered here. Second, equation (2) schedules job $i, \ldots, j_3$ consecutively in σ_A from $S_i(\pi^*) - k$ to $C_{j_3}(\pi^*) - k$, as prescribed by the definition of the value function, unless

$i = n + 1$. Note that the scheduling costs of jobs $i, \ldots, j_3$ are already added in the boundary condition and thus no scheduling cost is added by this equation. The feasibility and sufficiency of this equation is verified by Lemma 2. Third, equation (3) schedules jobs j in σ_A, under the condition that its start time $t - p_j$ is nonnegative, its time disruption is no more than k, and it is processed earlier than job i. Fourth, if conditions are not met for any of the three equations, then job j cannot be feasibly scheduled and a prohibitively large cost occurs.

Note that for each of equations (1-3) in the recurrence relation, the conditions of job j are necessary, but not sufficient. In other words, a job j may satisfy conditions for more than one equation, and the recurrence relation schedules j in a way that finds an optimal schedule.

Theorem 1 *Algorithm 1 finds an optimal schedule for problem* $1|a, \Delta_{\max} \leq k| \sum w_j C_j$ *in* $O(n^2 T_1)$ *time.*

Proof. The optimality follows from the fact that Algorithm 1 implicitly enumerates all the schedules satisfying the optimal properties in Lemmas 1–5, subject to the maximum disruption constraint. For time complexity, first note that the number of possible state values is $O(n^2 T_1)$ for the value function $f_i(j, t)$, because i and j are both in the order of n and t is in the order of T_1. Second, in the recurrence relation, each equation and its conditions require only a constant number of computations. Hence the complexity result holds. □

Next we provide a fully polynomial time approximation scheme (FPTAS) (Schuurman and Woeginger 2001), {Algorithm 1_ϵ}, for problem $1|a, \Delta_{\max} \leq k| \sum w_j C_j$. For any given $\epsilon > 0$, Algorithm 1_ϵ delivers a schedule σ^ϵ with $\sum w_j C_j(\sigma^\epsilon) / \sum w_j C_j(\sigma^*) \leq 1+\epsilon$, with time complexity polynomial in n and $1/\epsilon$. Algorithm 1_ϵ is based on Algorithm 1. However, in the dynamic program used in Algorithm 1_ϵ, we only refer to states instead of distinguishing between state variables and the value function. Also, a recurrence relation is not explicitly written. This is to increase flexibility that allows for suitable rounding techniques to be applied, which is necessary to achieve a time complexity polynomial in n and $1/\epsilon$.

Algorithm 1_ϵ

Input

Given $p_1, \ldots, p_n$, $w_1, \ldots, w_n$, $\pi^* = (1, \ldots, n)$ in which jobs are indexed by an SWPT indexing rule, T_1, T_2, k, and ϵ.

Initialization

Compute $A = \sum w_j C_j(\pi^*)$, $\delta = \epsilon A/n$, and $j_3 = \max\{j | C_j(\pi^*) - k \leq T_1\}$.

State

$(j, t, v)_i$ corresponds to a partial schedule containing jobs $\{1, \ldots, j\} \cup \{i, \ldots, j_3\}$ where (a) jobs $i, \ldots, j_3$ occupy the interval $[S_i(\pi^*) - k, C_{j_3}(\pi^*) - k]$; (b) the interval $[0, t]$ with $t \leq S_i(\pi^*) - k$ is occupied by jobs from $\{1, \ldots, \min\{j, i-1\}\}$ without idle time; (c) the total weighed completion time of the partial schedule is v.

Initial State

$(0, 0, \sum_{m=i}^{j_3} w_m(C_m(\pi^*) - k))_i$, for $S_i(\pi^*) - k > 0$ and $C_i(\pi^*) > T_2$.

Trial State Generation

For each state $(j, t, v)_i$, generate at most three trial states: 1) $(j+1, t + p_{j+1}, \min\{2A + 1, v + w_{j+1}(t + p_{j+1})\})_i$ if $j + 1 \leq i - 1$ and $C_{j+1}(\pi^*) - k \leq \min\{t + p_{j+1}, T_1\}$; 2) $(j_3 + 1, t, v)_i$ if $j + 1 \in \{i, \ldots, j_3\}$; 3) $(j + 1, t, \min\{2A + 1, v + w_{j+1}(T_2 + \sum_{m=1}^{j+1} p_m - t - \sum_{m=i}^{\min\{j+1, j_3\}} p_m)\})_i$, if $j + 1 \notin \{j_3 - i + 1, \ldots, j_3\}$ and $S_{j+1}(\pi^*) + k \geq T_2$.

Trial State Rounding

For each trial state $(j + 1, t, v')_i$, replace it with $(j + 1, t, \lfloor v'/\delta \rfloor \delta)_i$ if $v' \leq 2A$.

Rounded Trial State Elimination

For each pair of rounded trial states $(j + 1, t', v)_i$ and $(j + 1, t'', v)_i$ (where v is an integer multiple of δ), eliminate the second state if $t' \leq t''$, and eliminate the first state otherwise.

Termination Test

If $j + 1 < n$, then set $j = j + 1$ and return to the Trial State Generation step. Otherwise, select a state $(n, t, \hat{v})_i$ for which $\hat{v}$ is the minimum. If $\hat{v} \leq 2A$, then backtrack to find the corresponding schedule σ^ϵ; otherwise, Let $\delta = 2\epsilon A/n$ and $A = 2A$, and return to the Trial State Generation step to regenerate all the trial states from $j = 0$.

Let $B = \sum w_j C_j(\sigma^H) \geq \sum w_j C_j(\sigma^*)$, where σ^H is any feasible schedule of problem $1|a, \Delta_{\max} \leq k| \sum_j w_j C_j$. For example, σ^H can be found using Algorithm 2 introduced in Section 5.

Proposition 1 *The family of algorithms* $\{$*Algorithm* $1_\epsilon\}$*, for* $\epsilon > 0$*, is an FPTAS for problem* $1|a, \Delta_{\max} \leq k| \sum w_j C_j$*, with* $O((n^3 \log B)/\epsilon)$ *running time.*

Proof. Algorithm 1_ϵ constructs a schedule in the same way as Algorithm 1, and thus without the rounding of the Trial State Rounding step, can obtain an optimal schedule. Since no rounding is applied to the state variable t, the Trial State Generation step ensures feasibility of the schedule found.

We now analyze the total weighted completion time of the schedule delivered by Algorithm 1_ϵ. The Termination Test step selects the state $(n, t, \hat{v})_i$ and schedule σ^ϵ. First, since state variable v is always rounded down, we have $\hat{v} \leq \sum w_j C_j(\sigma^*)$. Also, in each execution of the Trial State Rounding step, replacing v' by $\lfloor v'/\delta \rfloor \delta$ decreases the value of the state variable from its true value by at most δ, and this computation is performed at most n times, thereby giving a total error of at most $n\delta$ due to this type of rounding. Hence, we have $\sum w_j C_j(\sigma^\epsilon) \leq \hat{v} + n\delta$. Finally, noting that $\sum w_j C_j(\sigma^*) \geq A$ in the Termination Test step, we have $\sum w_j C_j(\sigma^\epsilon) \leq \sum w_j C_j(\sigma^*) + n\delta \leq \sum w_j C_j(\sigma^*) + n(A\epsilon/n) \leq (1+\epsilon) \sum w_j C_j(\sigma^*)$.

Next we analyze the time complexity of Algorithm 1_ϵ. First, we compute the number of values of the state variables that remain after the Rounded Trial State Elimination step. States are generated for n values of j, and at most n values for i. Also, for each pair of fixed A and δ, after rounding, v is an integer multiple of δ and $v \leq 2A + 1$, which shows that the maximum number of values of v considered is $1 + (2A+1)/\delta = 1 + (2A+1)/(\epsilon A/n) = O(n/\epsilon)$. For each j and each rounded value of v, only a single value of t remains after the Rounded Trial State Elimination step. Thus, the overall number of state variable values that remain after the Rounded Trial State Elimination step is $O(n^3/\epsilon)$ for a fixed pair of A and δ. Noting that $B \geq \sum w_j C_j(\sigma^*)$, there is no need to consider $v > B$ in the Trial State Generation step, and the number of pairs of A and δ considered is $O(\log B)$. Since each value of these state variables generates at most three trial states, the Rounded Trial State Elimination step requires constant time for each trial state. Therefore, the overall time complexity of Algorithm 1_ϵ is $O((n^3 \log B)/\epsilon)$. □

Proposition 1 indicates that problem $1|a, |\sum w_j C_j$ also admits an FPTAS. Note that for problem $1|a|\sum w_j C_j$, there is no need to consider inserted idle time period, and it is straightforward to develop an optimal dynamic programming algorithm with time complexity $O(nT_1)$, similar to Algorithm 1 and a dynamic programming algorithm in Section 3.1.2 of Lee (1996). Correspondingly, an FPTAS with time complexity of $O((n^2 \log B)/\epsilon)$ can be

developed.

Corollary 1 *Problem* $1|a|\sum w_jC_j$ *admits an FPTAS with running time* $O((n^2 \log B)/\epsilon)$.

5 Allocation of Saving from Rescheduling

In this section we investigate how different job owners can allocate the *cost saving* from cooperation in rescheduling among themselves in a mutually beneficial and fair way. Since the initial schedule becomes infeasible after the machine unavailability, we first need to define a *naturally occurring feasible* schedule after the machine unavailability, relative to which a cost saving can be achieved by the cooperation of job owners in rescheduling. Note that the formation of a feasible schedule requires the cooperation of certain job owners. For example, we can move all the jobs completed later than T_1 in π^* backward, or move the jobs completed later than T_1 and started earlier than T_2 in π^* to the end of π^*. Determining which way is more natural is subjective, and we choose one approach that is meaningful and easy to implement in practice.

We consider the earlier mentioned natural way to react after the machine unavailability, which schedules each job as early as possible, following the SWPT indexing rule as in π^*. To be specific, we have:

Algorithm 2

Step 0. Given the job data: $\pi^* = (1, \ldots, n)$, T_1, T_2 and k.

Step 1. Find $j_1 = \min\{j | C_j(\pi^*) > T_1\}$.

Step 2. Schedule jobs $1, \ldots, j_1 - 1$ sequentially in the interval $[0, \sum_{j=1}^{j_1-1} p_j]$, and schedule jobs $j_1, \ldots, n$ sequentially in the interval $[T_2, T_2 + \sum_{j=j_1}^{n} p_j]$.

Proposition 2 *For problem* $1|a, \Delta_{\max} \leq k|\sum w_jC_j$, *Algorithm 2 has a tight worst-case performance ratio of 9/7 if* $w_1 = w_2 = \ldots = w_n$; *a tight worst-case performance ratio of 3 if* $T_2 - T_1 \leq \max_{j\in\{1,\ldots,j_1\}}\{p_j\}$; *and an unbounded worst-case performance ratio even if* $w_j = p_j$ *for* $j = 1, \ldots, n$.

Proof. First, note that the schedule obtained by Algorithm 2 has the minimum maximum time disruption among all feasible schedules. Thus, the worst-case performance ratio of Algorithm

2 for problem $1|a, \Delta_{\max} \leq k| \sum w_j C_j$ is no greater that for problem $1|a, | \sum w_j C_j$ (i.e., $k = \infty$). Then, the case with ratio 9/7 follows Lee and Liman (1992), with ratio 3 follows Kacem and Chu (2008), and with unbound ratio follows Lee (1996). □

We also need to define a time period within which a coalition can reschedule its jobs to minimize its scheduling cost, subject to the maximum time disruption constraint. If the completion time of a coalition is no less than T_1, then the machine unavailability does not affect the schedule of the coalition, and thus no rescheduling is needed and no cost saving occurs for the coalition. Otherwise, the machine unavailability affects the schedule of the coalition in a complicated way and it is necessary to define within what time period the coalition can schedule its jobs in rescheduling. In schedule σ^H found by Algorithm 2, each job is scheduled in a time period, based on which we define the time period a coalition can use to schedule its jobs.

Definition 1 *The* usable time period *of a coalition containing only jobs consecutively processed in schedule σ^H is from the start time of the first job, or from time 0 if job 1 is in the coalition, to the completion time of the last job in the coalition in schedule σ^H, minus any machine unavailable period within the time horizon.*

With schedule σ^H and usable time period specified for each coalition, a characteristic function v on the set of coalitions, can be defined, in which $v(\mathcal{S})$ corresponds to the cost saving from rescheduling its jobs in its usable time period relative to its total scheduling cost in schedule σ^H. Following the convention of single machine sequencing games (Curiel et al. 2002), we assume that two jobs can only switch their positions when both jobs and all the jobs processed between them in the original schedule π^*, or equivalently σ^H, are in the same coalition. This restriction is practically meaningful in that rescheduling of jobs non-consecutive in σ^H will cause at least sequence change of jobs scheduled between them and hence should not be allowed unless the jobs in between also cooperate, i.e., are in the same coalition. A coalition of jobs $j, \ldots, l$ processed in that sequence in σ^H is denoted $\mathcal{S}(j, l)$. Note that any coalition containing jobs non-consecutively processed in schedule σ^H can be split into several maximum sets of consecutively processed jobs, and its characteristic function value is equal to the sum of the characteristic function values of these sets of jobs.

For the rescheduling game defined via $v(\mathcal{S})$, our focus is the *core* (Gillies 1953) and *Shapley*

value (Shapley 1953), two central solutions regarding value allocation in cooperative game theory. A core allocation distributes the value generated by all the cooperating players such that no coalition can achieve more value by itself, and thus cooperation is encouraged. The set of all core allocations is the core of the game. A game with a nonempty core for all instances is *balanced*. The Shapley value of a cooperative game allocates each player its average marginal contribution, over all the permutations of all the players by which the grand coalition is formed. The two concepts are essentially different: while core allocation incentivizes the stability of the grand coalition, Shapley value incorporates axiomatic fairness regarding marginal contribution of each player. Also, core allocation may not exist or unique, while Shapley value uniquely exists. Note that the Shapley value may not be in the core of a game instance, even if the game is balanced. As for our rescheduling game, core allocation exists and may not be unique, which motivates us to further consider the Shapley value.

Under the definition of usable time period for each coalition, it is not difficult to see, if a coalition does not contain job $j_1 - 1$ or j_1, then the coalition does not need to reschedule its jobs; simply processing its jobs in the SWPT order as in σ^H in the usable time period minimizes the scheduling cost of the coalition. If a coalition contains only consecutively processed jobs in σ^H and both job $j_1 - 1$ and job j_1, then an optimal schedule of the coalition can be found by Algorithm 1. Note that the schedule found by Algorithm 1 should only use the usable time period of a coalition. It is not obvious that Algorithm 1 can always find a feasible schedule for each coalition using the coalition's usable time period, since the usable time period enforces a deadline for the makespan. Next Lemma 7 shows that Algorithm 1 indeed guarantees feasibility for each coalition.

Lemma 7 *For any instance of problem* $1|a, \Delta_{\max} \leq k|\sum w_j C_j$*, the maximum completion time of the schedule* σ^* *found by Algorithm 1 is no more than that of the schedule* σ^H *found by Algorithm 2.*

Proof. Since no machine idle time exists between the processing of any two jobs for both schedules σ^* and σ^H after time T_2, it is sufficient to show that the total processing time of jobs of partial schedule σ_A^* is no less than that of partial schedule σ_A^H. By contradiction, assume that the total processing time of jobs in σ_A^* is strictly less than that of σ_A^H.

Note that for both schedules σ^* and σ^H, jobs before and after the machine disruption period are processed in SWPT order, respectively. Also, jobs in partial schedule σ_A^H, denoted J_1, are of the smallest SWPT indexes among all the jobs. Therefore, in σ^*, jobs in J_1 are processed at the beginning parts of partial schedules σ_A^* and σ_B^*, respectively. We adjust schedule σ^* as follows: move the jobs in J_1 from partial schedule σ_B^* to be processed immediately after the jobs in J_1 in partial schedule σ_A^*, in SWPT order and each as early as possible; move the jobs not in J_1 in partial schedule σ_A^* to be processed from time T_2 in SWPT order and each as early as possible; the schedule of other jobs, i.e., jobs processed at the end of σ_B^H, remains the same. It is straightforward to verify, under the assumption that the total processing time of jobs in σ_A^* is strictly less than that of σ_A^H, the schedule obtained after such adjustments is feasible and of a total scheduling cost strictly less than that of σ^*, which contradicts the optimality of σ^*. □

Lemma 7 is intuitive to follow. Generally, Algorithm 1 is more efficient than Algorithm 2 since it more efficiently uses machine time before time T_1, and thus leaves less idle time before T_1 and consequently finds a schedule with makespan less than an inferior schedule found by Algorithm 2. In fact, in Example 1, schedule σ^* has a makespan of 17, while schedule σ^H has a makespan of 18.

Recall that z^* is the cost of an optimal schedule σ^* found by Algorithm 1. Obviously, $z^* \leq z^H$. Consider the following payoff vector $\mathbf{x} = (x_1, \ldots, x_n)$ for players $1, \ldots, n$.

Payoff vector x

$x_j = 0$, for $j = 1, \ldots, j_1 - 2, j_1 + 1, \ldots, n$;

$x_{j_1-1} = \delta(z^H - z^*)$;

$x_{j_1} = (1 - \delta)(z^H - z^*)$, where δ can be any value between 0 and 1.

Theorem 2 *The vector* $\mathbf{x}$ *is a core allocation for the rescheduling game.*

Proof. First, it is straightforward to verify the efficiency that $\sum_{i=1}^{n} x_i = z^H - z^*$. Second, for the coalitional rationality, note that any coalition not containing player $j_1 - 1$ or j_1 cannot be better off by acting by itself under the definition of usable time period of each coalition. If a coalition contains both players $j_1 - 1$ and j_1, then its maximum cost saving by rescheduling from schedule σ^H is $z^H - z^*$, which is achieved from vector $\mathbf{x}$. □

Note that the rescheduling game is superadditive and belongs to a more general class of σ_0-*component additive* game (Curiel et al. 1994), where players are ordered and the characteristic function value of a coalition is additive over values of the split of maximum sub-coalitions containing only players in consecutive order. Curiel et al. (1994) provide a description of core allocation, namely β-*rule*, for any σ_0-component additive game. Below we apply the the β-rule to our game.

Payoff vector $\mathbf{y}^\delta$ defined by β-rule

$y_j^\delta = \delta(v(\mathcal{S}(1,j)) - v(\mathcal{S}(1,j-1)) + (1-\delta)(v(\mathcal{S}(j,n)) - v(\mathcal{S}(j+1,n)))$, where $0 \leq \delta \leq 1$.

Remark 1 *The vector $\mathbf{y}^\delta$ is a core allocation for the rescheduling game.*

Slikker (2006) studies a relaxed version of the standard one machine sequencing game in that rescheduling of jobs in any coalition is allowed as long as it does not delay the processing of any job outside the coalition. Following the same idea, we can define a *relaxed rescheduling game* that allows rescheduling of the jobs in a coalition if no jobs outside the coalition are delayed, without introducing the definition of the usable time period. However, the relaxed rescheduling game is not balanced, as shown by the following example.

Example 2 $n = 4$; $p_1 = 12$, $p_2 = p_3 = p_4 = 3$; $w_1 = 12$, $w_2 = w_3 = w_4 = 2$; $T_1 = 6$, $T_2 = 12$; $k = 24$. In schedule σ^H, jobs are scheduled with completion times of 24, 27, 30 and 33 for jobs 1, 2, 3 and 4, respectively. For coalition $\{2,3\}$, the optimal schedule after rescheduling is to have jobs 2 and 3 completed at times 3 and 6, respectively, achieving a cost saving of 96, i.e., $v(\{2,3\}) = 96$. For coalition $\{2,4\}$, the optimal schedule after rescheduling is to have jobs 2 and 4 completed at times 3 and 6, respectively, achieving a cost saving of 102, i.e., $v(\{2,4\}) = 102$. For coalition $\{3,4\}$, the optimal schedule after rescheduling is to have jobs 3 and 4 completed at times 3 and 6, respectively, achieving a cost saving of 108, i.e., $v(\{3,4\}) = 108$. For the instance to have a core allocation, the cost saving of coalition $\{2,3,4\}$ needs to be no less than $(96 + 102 + 108)/2 = 153$. However, for coalition $\{2,3,4\}$, the optimal schedule after rescheduling is to have jobs 2, 3 and 4 completed at times 3, 6 and 27, respectively, achieving a cost saving of only 108, i.e., $v(\{2,3,4\}) = 108 < 153$.

Next we consider the Shapley value of the rescheduling game. Let $\Pi_{\mathcal{N}}$ be the set of all the $n!$ permutations of the n jobs in the grand coalition, and $\pi \in \Pi_{\mathcal{N}}$ be any permutation. Let $\mathcal{P}(\pi, i)$ be the set of jobs preceding job i in permutation π. The Shapley value, a payoff vector $\phi(v)$ for the n players of a cooperative game with players $\mathcal{N}$, is defined as follows:

$$\phi_i(v) = \frac{1}{n!} \sum_{\pi \in \Pi_{\mathcal{N}}} (v(\mathcal{P}(\pi, i) \cup \{i\}) - v(\mathcal{P}(\pi, i))). \tag{5}$$

For a general cooperative game, it is difficult to compute $\phi_i(v)$ defined in (5). But for the rescheduling game, since two jobs can only switch their positions when both jobs and all the jobs processed between them in schedule σ^H are in the same coalition, we are able to find the Shapley value in a relatively easy way. In fact, the special coalition structure of the game allows an easier determination of *unanimity coefficients*, i.e., the unique coefficients attached to *unanimity games* of the rescheduling game that form a basis. Then, these coefficients can be used to describe the Shapley value of the game (Shapley 1953). Such an approach, for example, is used by Borm et al. (2002) and Maniquet (2003). Borm et al. (2002) obtain expression for the value of unanimity coefficients for an arbitrary σ_0-component additive game, which can be used to obtain the Shapley of the rescheduling game we consider. Maniquet (2003) describes the Shapley of a single machine sequencing game with equal processing time and with no initial job sequence. Maniquet (2003) mentions that there are several ways of proving this lemma when defining the Shapley value for their game and chooses to use the way with unanimity coefficients. Thus, a direct construction of the Shapley value for the rescheduling game would provide readers an extra layer of information. Below we use a direct method to find the Shapley value of the rescheduling game, using only definition (5). We consider the following four cases regarding how a player i can actually contribute to a coalition formed in a permutation π.

Case 1. Consider a coalition $\mathcal{S}(j, l)$ that contains jobs $j, \ldots, l$ processed consecutively in that sequence in σ^H, where $1 < j \leq j_1 - 1$ and $j_1 \leq l < n$ such that rescheduling of jobs $j, \ldots, l$ is potentially beneficial to the coalition. The cases where $j = 1$ or $l = n$ will be discussed separately. The marginal contribution of job i, $j \leq i \leq l$, to coalition $\mathcal{S}(j, l)$ is $v(\mathcal{S}(j, l)) - v(\mathcal{S}(j, l) \setminus \{i\})$, which can be computed through the use of Algorithm 1. Note that two jobs can only switch their positions when both jobs and all the jobs processed between them in schedule σ^H are in the same coalition. Also, any set of jobs not containing

job $j_1 - 1$ or j_1 cannot achieve cost saving through rescheduling. Therefore, the marginal contribution of job i to any coalition that contains jobs $j, \ldots, l$ but not $j-1$ or $l+1$ is also $v(\mathcal{S}(j,l)) - v(\mathcal{S}(j,l) \setminus \{i\})$. This provides us basis to compute the number of permutations by which the marginal contribution of job i is equal to $v(\mathcal{S}(j,l)) - v(\mathcal{S}(j,l) \setminus \{i\})$. For each of these permutations, we need to have:

1. Jobs in $\mathcal{S}(j,l) \setminus \{i\}$ can be in any order, in a total a $(l-j)!$ orders;
2. Job i must be after jobs in $\mathcal{S}(j,l) \setminus \{i\}$;
3. Jobs $j-1$ and $l+1$ must be after job i, in 2 different orders;
4. Jobs $1, \ldots, j-2, l+2, \ldots, n$ can be in any order among jobs $j-1, \ldots, l+1$, in a total of $(l-j+4)(l-j+5)\ldots(n)$ orders.

Thus the total number of considered permutations with a marginal contribution of $v(\mathcal{S}(j,l)) - v(\mathcal{S}(j,l) \setminus \{i\})$ from job i is

$$2(l-j)!(l-j+4)(l-j+5)\ldots(n),$$

which is a fraction

$$\frac{2}{(l-j+1)(l-j+2)(l-j+3)} \tag{6}$$

of the total number of all permutations $n!$.

Case 2. This case is similar to Case 1 except that $j = 1$ and $l < n$. Following analysis similar for Case 1, we have that the total number of considered permutations with a marginal contribution of $v(\mathcal{S}(1,l)) - v(\mathcal{S}(1,l) \setminus \{i\})$ from job i, $1 \le i \le l$ and $j_1 \le l$, is $(l-1)!(l+2)(l+3)\ldots(n)$, which is a fraction $1/(l(l+1))$ of the total number of all permutations $n!$.

Case 3. This case is similar to Case 1 except that $j > 1$ and $l = n$. Following analysis similar for Case 1, we have that the total number of considered permutations with a marginal contribution of $v(\mathcal{S}(j,n)) - v(\mathcal{S}(j,n) \setminus \{i\})$ from job i, $j \le i$ and $1 < j \le j_1 - 1$, is $(n-j)!(n-j+3)(n-j+4)\ldots(n)$, which is a fraction $1/((n-j+1)(n-j+2))$ of the total number of all permutations $n!$.

Case 4. This case is similar to Case 1 except that $j = 1$ and $l = n$. It is easy to see, the total number of considered permutations with a marginal contribution of $v(\mathcal{S}(1,n)) - v(\mathcal{S}(1,n) \setminus \{i\})$ from any job i is $(n-1)!$, which is a fraction $1/n$ of the total number of all permutations $n!$.

Note that Cases 1–4 consider all the sets of consecutively processed jobs in schedule σ^H over which a job i within the set can have nonzero marginal contribution, and thus no other cases are needed. Combining Cases 1–4, we can find the Shapley value of the rescheduling game. For simplicity, we denote $v_{-i}(j,l) = (\mathcal{S}(j,l)) - v(\mathcal{S}(j,l) \setminus \{i\})$.

Theorem 3 *The Shapley value $\phi(v)$ of the rescheduling game is given by:*

$$\begin{aligned}\phi_i(v) &= \sum_{1<j\le j_1-1, j_1\le l<n, j\le i\le l} \frac{2v_{-i}(j,l)}{(l-j+1)(l-j+2)(l-j+3)} \\ &+ \sum_{j_1\le l<n, i\le l} \frac{v_{-i}(1,l)}{l(l+1)} + \sum_{1<j\le j_1-1, j\le i} \frac{v_{-i}(j,n)}{(n-j+1)(n-j+2)} + \frac{v_{-i}(1,n)}{n}, \end{aligned} \tag{7}$$

which can be found in $O(n^3T_1)$ time.

Proof. Equation (7) defines the Shapley value following the arguments immediately preceding the Theorem. For the complexity, note that it is the same as computing all the characteristic function values of coalitions in the form of $\mathcal{S}(j,l)$, for any $1 \le j < l \le n$. Algorithm 1 varies l from 1 to n, but fixes $j = 1$. Hence an extra level of $O(n)$ time is needed. Therefore, the overall complexity to calcuate the Shapley value is $O(n^3T_1)$. □

Note that the computation of the Shapley value defined by (7) involves the implementation of Algorithm 1 for coalitions containing only jobs consecutively processed in σ^H, including both job $j_1 - 1$ and job j_1, and thus runs in a pseudopolynomial time. The Shapley value is not necessarily a core allocation, as shown by Example 3.

Example 3 $n = 4$; $p_1 = 1$, $p_2 = 1$, $p_3 = 3$, $p_4 = 1$; $w_1 = 1$, $w_2 = 1$, $w_3 = 3$, $w_4 = 1$; $T_1 = 3$, $T_2 = 4$; $k = 5$. Schedule π^* processes the jobs in the sequence of 1, 2, 3, and 4. It is easy to verify, $v(\mathcal{S}(1,4)) = v(\mathcal{S}(2,4)) = 5$, and hence in any core allocation the payoff to player 1 must be 0. However, we have $v(\mathcal{S}(1,3)) = 4$ and $v(\mathcal{S}(2,3)) = 0$. That is, job 1 has marginal contribution of 4 to jobs 2 and 3, and hence must receive a nonzero payoff from the Shapley value. In fact, we have that the Shapley value is $\{1/3, 2, 2, 2/3\}$, which is not a core allocation.

The Shapley value is not necessarily in the core. A sufficient condition for the Shapley value to be in the core is that the game is *convex*. A game is convex if for any coalitions $\mathcal{S}_1 \subseteq \mathcal{S}_2$ and any player $i \notin \mathcal{S}_1$, we have $v(\mathcal{S}_1 \cup \{i\}) - v(\mathcal{S}_1) \le v(\mathcal{S}_2 \cup \{i\}) - v(\mathcal{S}_2)$. That is, the marginal contribution of a player to a coalition is nondecreasing when the coalition enlarges.

For our rescheduling game, it is straightforward to see that as a coalition enlarges, the marginal contribution of a player can be deceasing, because the machine idle time immediately preceding T_1 in schedule σ^H could be used by other players in the enlarged coalition and thus the player cannot further achieve cost saving, even in case all the jobs are with the same processing time and weight. However, when the maximum allowable time disruption k is sufficiently small, then the game is convex and consequently the Shapley value is in the core.

Remark 2 *Suppose job j is the first processed job in schedule σ^H with $S_j(\sigma^H) \geq T_2$, $p_j \leq T_1 - C_{j_1-1}(\sigma^H)$ and $C_j(\sigma^H) - T_1 \leq k$. If $T_2 - S_{j_1-1}(\sigma^H) > k$ and $C_{j+1}(\sigma^H) - T_1 > k$, then the rescheduling game is convex.*

Proof. First, if a coalition does not contain jobs $j_1 - 1, j_1, \ldots, j$, then no cost saving can be achieved. Second, it is easy to verify, for any coalition containing jobs $j_1 - 1, j_1, \ldots, j$, the cost saving is achieved by scheduling job j to compete at $C_j(\sigma^H) - k \leq T_1$, and processing jobs after job j in schedule σ^H to complete earlier by p_j, and consequently, the contribution to the saving by any job in $j_1 - 1, j_1, \ldots, j$ is nondecreasing as the coalition enlarges. Any other job, among jobs $1, \ldots, j_1 - 2, j + 1, \ldots, n$, if in the coalition, also contributes to the cost saving nondecreasingly as the coalition grows. Therefore, the game is convex. □

In our computational studies in Section 6, we show that the Shapley value is likely a core allocation for random instances.

6 Computational Studies

The purposes of our computational studies are threefold. First, we evaluate how the *percentage rescheduling cost*, the percentage of extra cost caused by rescheduling using Algorithm 1 relative to the scheduling cost of the initial schedule π^*, i.e., $100(z^* - y^*)/y^*$, is affected by the machine unavailability. Second, we study how the *percentage cost saving*, the cost saving in percentage achieved by rescheduling using Algorithm 1 compared with using Algorithm 2, i.e., $100(z^H - z^*)/z^H$, is affected by the machine unavailability. Third, we investigate how the above mentioned cost saving $z^H - z^*$ is distributed to different job owners using the Shapley value defined by (7).

Now we describe how to generate a set of random problem instances in our studies. We follow the guidelines proposed by Hall and Posner (2001) to create a broad range

of parameter specifications that could affect the analysis but can be rescaled without significantly affecting their effects. We randomly generate 1,000 problem instances for each combination of the specifications for four parameters, n, T_1, $D = T_2 - T_1$, and k. First, we randomly generate $p_j \sim UI[1, \ldots, 100]$ and $w_j \sim UI[1, \ldots, 100]$. Second, we use $n \in \{20,$ 40, 60, 80, 100, 150, 200$\}$. Third, we have $T_1 \in \{\lfloor P/4 \rfloor, \lfloor P/2 \rfloor, \lfloor 3P/4 \rfloor\}$. Fourth, we consider $D \in \{\lfloor P/50 \rfloor, \lfloor P/25 \rfloor, \lfloor P/10 \rfloor\}$. Finally, for the specifications for k, we have $k \in \{D+100, D+\lfloor 2.5P/n \rfloor, D+\lfloor 3P/n \rfloor, D+\lfloor 3.5P/n \rfloor, D+\lfloor 4P/n \rfloor\}$. Overall, $7*3*3*5 = 315$ combinations are considered.

Table 1: Effect of Parameters on Percentages of Rescheduling Cost and Cost Saving

Parameter	APO	MPO	APE	MPE
$n = 20$	5.52%	31.92%	1.77%	13.70%
$n = 40$	4.82%	23.63%	1.04%	7.72%
$n = 60$	4.67%	21.23%	0.79%	5.24%
$n = 80$	4.59%	18.66%	0.64%	3.85%
$n = 100$	4.55%	19.14%	0.52%	2.84%
$n = 150$	4.51%	17.41%	0.36%	2.13%
$n = 200$	4.50%	17.10%	0.28%	1.48%
$T_1 = \lfloor P/4 \rfloor$	8.87%	31.92%	1.33%	13.70%
$T_1 = \lfloor P/2 \rfloor$	4.17%	17.76%	0.75%	9.69%
$T_1 = \lfloor 3P/4 \rfloor$	1.18%	7.82%	0.23%	4.90%
$D = \lfloor P/50 \rfloor$	2.01%	17.77%	0.69%	13.18%
$D = \lfloor P/25 \rfloor$	3.63%	21.24%	0.75%	13.32%
$D = \lfloor P/10 \rfloor$	8.57%	31.92%	0.87%	13.70%
$k = D + 100$	4.92%	31.92%	0.60%	12.48%
$k = D + \lfloor 2.5P/n \rfloor$	4.79%	28.11%	0.72%	13.19%
$k = D + \lfloor 3P/n \rfloor$	4.71%	28.04%	0.80%	13.70%
$k = D + \lfloor 3.5P/n \rfloor$	4.65%	23.80%	0.85%	13.70%
$k = D + \lfloor 4P/n \rfloor$	4.62%	23.80%	0.88%	13.70%
Overall	4.74%	31.92%	0.77%	13.70%

Table 1 summarizes the computational results regarding the percentages of rescheduling cost and cost saving, where the first column contains values of the input parameters n, T_1, D, and k; column APO (resp., MPO) contains the average (resp., maximum) percentage increment of the cost of an optimal schedule after the machine unavailability relative to that of the original schedule; and column APE (resp., MPE) contains the average (resp., maximum) percentage reduction of the cost of the optimal schedule found by Algorithm 1 relative to that

of the schedule found by Algorithm 2. We have the following findings from Table 1 about how percentages of rescheduling cost and cost saving are affected by different problem parameters.

First we consider how the percentage of rescheduling cost changes. First, the percentage rescheduling cost decreases with the number of jobs n, but such an impact becomes dimmish when n is large, such as no less than 100. This may because that the cost of the initial optimal schedule π^* increases with n faster than the extra cost caused by machine unavailability, especially when n is relatively small, where the same increment over a smaller n constitutes a larger portion of n. Second, a larger impact on the percentage rescheduling cost occurs when the machine unavailable period starts earlier. This is because the scheduling cost considered, the total weighted completion time, is increasing with the completion time of each job and is also accumulative over all the jobs. Finally, the impacts of the duration of the machine unavailable period and the maximum allowable disruption are intuitive, and we summarize and explain the results in the appendices.

Second we discuss how the percentage of cost saving changes. First, the percentage cost saving decreases with the number of jobs n. Note that a larger number of n might provide more space for Algorithm 1 to improve a heuristic schedule, but a larger n is also likely to increase the cost of the schedule found by Algorithm 2. We consider the cost saving relative to the cost of schedule found by Algorithm 2, and the percentage cost saving decreases with n on average. Second, the percentage cost saving decreases with the start time of the machine unavailability. This relationship can be explained by the intuition that a smaller value of T_1 would provide a larger space for Algorithm 1 to improve the heuristic schedule found by Algorithm 2. In addition, note that the effects of factors n and T_1 on the percentage of cost saving are in the same pattern as on the percentage rescheduling cost, but are in a smaller magnitude (0.77% versus 4.74% on average). Finally, the more intuitive impacts of D and k are provided in the appendices.

Next we study the distribution of the Shapley value to different job owners. For a clear demonstration, we focus on the cases with $n = 20$, and the machine unavailability starting exactly from the $C_{10} + \lfloor p_{11}/2 \rfloor$ that disrupts the processing of job 11. Thus, we have that $j_1 - 1 = 10$ and $j_1 = 11$. We vary parameters D and k the same as in our earlier studies, and generate 2,000 instances for each of the $3 * 5 = 15$ parameter combinations. For all the

30,000 instances tested, there are 18,617 with nonzero cost saving from the grand coalition, which are the ones we report. Among these 18,617 instances, for only 1,325 instances, 7.12% over the total, the Shapley value is not an element of the core. Note that to check whether the Shapley value is in the core, it suffices to check whether the total payoff received from the Shapley value is no less than the characteristic function value of coalitions containing only consecutively processed jobs in σ^H including both jobs $j_1 - 1$ and j_1.

Figure 2 depicts how the percentage of the total cost saving from rescheduling of jobs in the grand coalition is distributed among all the 20 job owners, averaged over the 18,617 instances with nonzero cost saving for the grand coalition. Note that for all the 18,617 instances, jobs 1 and 2 receive zero payoff from the Shapley value. Jobs 10 and 11 receive the same highest allocation percentage 21.6%, because the two jobs are necessary to be included by any coalition to achieve cost saving. The allocation increases as jobs are closer to job 10, and then decreases as the jobs are away from job 11. Jobs after job 11 receive more allocation from the Shapley value than jobs before job 10, essentially because they are more affected by the machine unavailability. Note that in general jobs processed significantly earlier than T_1 in the initial optimal schedule σ^* are less likely to be affected by the machine disruption. In the schedule σ^H found by Algorithm 2, their costs are the same as in schedule σ^*. Therefore, the Shapley value does not provide much payoff to owners of these jobs.

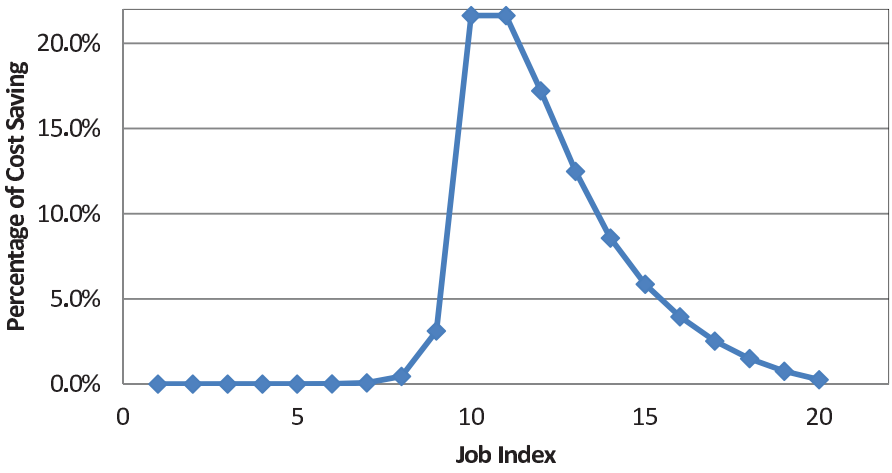

Figure 2: Average Percentage Allocations of Shapley Value

Table 2 shows how the average percentage sharing of the total cost saving from the grand

coalition to job owners 7–16 are affected by parameters D and k. When the duration of machine unavailability becomes longer, jobs 10 and 11 receive slightly more allocation, while other jobs receive slightly less allocation. This is because that with longer machine disruption, it is harder to reschedule jobs with longer distance from jobs 10 and 11 to save cost. When the allowable disruption becomes larger, jobs 10–13 obtain less allocation, while other jobs receive more allocation. This is because a larger value of k provides opportunity for jobs with larger distance from jobs 10 and 11 to play a more important role in cost saving from rescheduling, and thus increases their allocations accordingly.

Table 2: Effect of Parameters on Percentage Allocations of Shapley Value

Parameter	job 7	job 8	job 9	job 10	job 11	job 12	job 13	job 14	job 15	job 16
$D = \lfloor P/50 \rfloor$	0.07%	0.48%	3.30%	21.09%	21.09%	17.34%	12.61%	8.75%	6.06%	4.10%
$D = \lfloor P/25 \rfloor$	0.07%	0.46%	3.15%	21.42%	21.42%	17.26%	12.54%	8.60%	5.93%	4.02%
$D = \lfloor P/10 \rfloor$	0.07%	0.43%	2.90%	22.34%	22.34%	17.04%	12.29%	8.36%	5.59%	3.74%
$k = D + 100$	0.01%	0.07%	1.32%	22.55%	22.55%	18.25%	12.52%	8.35%	5.71%	3.83%
$k = D + \lfloor 2.5P/n \rfloor$	0.01%	0.10%	2.01%	22.22%	22.22%	17.54%	12.55%	8.53%	5.85%	3.94%
$k = D + \lfloor 3P/n \rfloor$	0.03%	0.33%	3.10%	21.64%	21.64%	17.17%	12.54%	8.58%	5.89%	3.99%
$k = D + \lfloor 3.5P/n \rfloor$	0.08%	0.59%	3.88%	21.25%	21.25%	16.90%	12.48%	8.64%	5.90%	3.97%
$k = D + \lfloor 4P/n \rfloor$	0.19%	0.96%	4.34%	20.95%	20.95%	16.67%	12.35%	8.65%	5.89%	3.97%
Overall	0.07%	0.45%	3.11%	21.63%	21.63%	17.21%	12.48%	8.57%	5.86%	3.95%

7 Conclusions

In this work we investigate a single machine rescheduling problem, where jobs belong to different owners. The jobs have been initially scheduled to minimize the total weighted completion time. Then the machine will be unavailable for a period of time during the planning horizon, and hence the initial schedule becomes infeasible. We restrict the disruption of a new schedule over the initial schedule such that the difference between the completion times of the two schedules of any job cannot be more than a given integer. A natural reaction to the machine unavailability is to process the jobs in the initial sequence, each as early as possible. Such an obtained schedule has the minimum maximum time disruption among all feasible schedules, and hence is feasible, but often its total scheduling cost can be reduced. With some specific definition, we define the time period over which a set of jobs can reschedule its jobs to achieve cost saving, and then define a cooperative game accordingly.

The rescheduling problem faced by the grand coalition is computationally intractable.

We develop an optimal pseudopolynomial time dynamic programming algorithm for the rescheduling problem, using several properties identified for an optimal schedule. Then we provide a core allocation of the total cost saving for the grand coalition, in a very simple form. We also find the Shapley value of the cooperative game in a form that can be computed in a reasonable amount of time. The Shapley value also applies to a broad range of single machine sequencing games. Then over a large set of randomly generated instances, we computationally evaluate the extra cost caused by machine unavailability, the cost saving from optimization relative to the naturally constructed schedule, the likelihood for the Shapley value to be a core allocation, and the pattern the Shapley value allocates the cost saving to different job owners.

Managerial insights are derived from extensive computational studies. First, the extra schedule cost caused by machine disruption is relatively large when the disruption occurs earlier with longer duration, and the allowable time disruption in rescheduling is small. Second, relative to a simple heuristic rule that processes each job as early as possible in the same sequence as the initial schedule, optimal rescheduling is more valuable when the number of jobs is large, the machine disruption occurs earlier with longer duration, and the allowable time disruption in rescheduling is large. Third, the Shapley value, not necessarily, but often provides incentives for the jobs owners to cooperate in an optimal rescheduling. Fourth, the Shapley value allocates more cost saving from optimal rescheduling to jobs processed close to or slightly later than the time period during which a machine disruption occurs.

Our work explicitly incorporates two classic scheduling topics: sequencing game and rescheduling. We believe our work primarily contributes to the connection between these two areas, and will arouse more research interests in further explorations of the integration between them, for example, considering other scheduling costs, other types of disruptions, and other solution concepts in cooperative game theory. It would be specifically interesting to thoroughly study problem $1|a^*, \Delta_{\max} \leq k| \sum w_j C_j$, as introduced in Section 3.

Acknowledgments

The authors are grateful to the two anonymous reviewers for their constructive comments, which substantially improved the quality of this work.

References

Adiri I., J. Bruno, E. Frostig, A.H.G. Rinnooy Kan. 1989. Single machine flow-time scheduling with a single breakdown. *Acta Informatica* **26** 679–696.

Agnetis, A., N.G. Hall, D. Pacciarelli. 2006. Supply chain scheduling: Sequence coordination. *Discrete Applied Mathematics* **154** 2044–2063.

Aydinliyim, T., X. Cai, G.L. Vairaktarakis. 2014. Capacity allocation and coordination issues for the timely processing of outsourced operations. *Journal of Systems Science and Systems Engineering* **23** 300–12.

Aydinliyim, T., G.L. Vairaktarakis. 2010. Coordination of outsourced operations to minimize weighted flow time and capacity booking costs. *Manufacturing & Service Operations Management* **12** 236–255.

Aydinliyim, T., G.L. Vairaktarakis. 2011. Sequencing strategies and coordination issues in outsourcing and subcontracting operations. In: *Planning Production and Inventories in the Extended Enterprise, International Series in Operations Research and Management Science*, K. Kempf, P. Keskinocak, R. Uzsoy (Eds), Springer, New York, NY, **151** pp. 269–319.

Aydinliyim, T., G.L. Vairaktarakis. 2013. A cooperative savings game approach to a time sensitive capacity allocation and scheduling problem. *Decision Sciences* **44** 357–376.

Azizoğlu, M., O. Alagöz. 2005. Parallel-machine rescheduling with machine disruptions. *IIE Transactions* **37** 1113–1118.

Aytug, H., M.A. Lawley, K. McKay, S. Mohan, R. Uzsoy. 2004. 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., J.R. Birge, J. Mittenthal, C.E. Noon. 1991. Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research* **39** 470–483.

Borm, P., G. Fiestras-Janeiro, H. Hamers, E. Sánchez, M. Voorneveld. 2002. On the convexity of games corresponding to sequencing situations with due dates. *European Journal of Operational Research* **136** 616–634.

Cai, X., G.L. Vairaktarakis. 2012. Coordination of outsourced operations at a third-party facility subject to booking, overtime, and tardiness costs. *Operations Research* **60** 1436–1450.

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

Curiel, I., H. Hamers, F. Klijn. 2002. Sequencing games: A survey. In: *Chapters in Game Theory: In Honor of Stef Tijs*. Eds.: P. Borm, H. Peters. Kluwer Academic Publishers, Boston, MA, 27–50.

Curiel, I.J., G. Pederzoli, S. Tijs. 1989. Sequencing games. *European Journal of Operational Research* **40** 344–351.

Curiel, I.J., J. Potters, R. Prasad, S. Tijs, B. Veltman. 1994. Sequencing and cooperation. *Operations Research* **42** 566–568.

Dawande, M., H.N. Geismar, N.G. Hall, C. Sriskandarajah. 2006. Supply chain scheduling: Distribution systems. *Production and Operations Management* **15** 243–261.

Garey, M.R., D.S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco.

Gillies, D.B. 1959. Solutions to general non-zero-sum games. In *Contributions to the Theory of Games*. Eds.: A.W. Tucker, R.D. Luce, **4** 47–85. Princeton University Press, Princeton, N.J.

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

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

Hall, N.G., Z. Liu. 2008. Cooperative and noncooperative games for capacity planning and scheduling. In: *Tutorials in Operations Research*, Z. Chen, S. Raghavan (Eds.), INFORMS, pp. 108–129.

Hall, N.G., Z. Liu. 2010. Capacity allocation and scheduling in supply chains. *Operations Research* **58** 1711–1725.

Hall, N.G, M.E. Posner. 2001. Generating experimental data for computational testing with machine scheduling applications. *Operations Research* **49** 854–865.

Hall, N.G., C.N. Potts. 2003. Supply chain scheduling: Batching and delivery. *Operations Research* **51** 566–584.

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

Hall, N.G., C.N. Potts. 2010. Rescheduling for job unavailability. *Operations Research* **58** 746–755.

Huo, Y., B. Reznichenko, H. Zhao, 2014. Minimizing total weighted completion time with unexpected machine unavailability. *Journal of Scheduling* **17** 161–172.

Jackson, J.R. 1955. Scheduling a production line to minimize maximum tardiness. Research Report 43, Management Science Research Project, University of California, Los Angeles, CA.

Kacem, I., C. Chu. 2008. Worst-case analysis of the WSPT and MWSPT rules for single machine scheduling with one planned setup period. *European Journal of Operational Research* **187** 1080–1089.

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

Lee, C.-Y., S.D. Liman. 1992. Single machine flow-time scheduling with scheduled maintenance. *Acta Informatica* **29** 375–382.

Li, C., X. Qi, C.-Y. Lee. 2015. Disruption recovery in liner shipping. *Transportation Science* **49** 900–921.

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

Maniquet, F. 2003. A characterization of the Shapley value in queueing problems. *Journal of Economic Theory* **109** 90–103.

Manoj, U.V., J.N.D. Gupta, S.K. Gupta, C. Sriskandarajah. 2008. Supply chain scheduling: Just-in-time environment. *Annals of Operations Research* **161** 53–86.

Manoj, U.V., C. Sriskandarajah, E. Wagneur. 2012. Coordination in a two-stage production system: Complexity, conflict and cooperation. *Computers & Operations Research* **39** 1245-1256.

Pinedo, M.L. 2012. *Scheduling, Theory, Algorithms and Systems*, 4th edition. Springer, New York, NY.

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

Schuurman, P., G.J. Woeginger. 2001. Approximation schemes: A tutorial. In *Lectures on Scheduling*, R.H. Möhring, C.N. Potts, A.S. Schulz, G.J. Woeginger, L.A. Wolsey (Eds.), Springer, Berlin.

Selvarajah, E., G. Steiner. 2008. Approximation algorithms for the supplier's supply chain scheduling problem to minimize delivery and inventory holding costs. *Operations Research* **57** 426–438.

Shapley, L.S. 1953. A value for n-person games. *Annals of Mathematics Study* **28** 307–317.

Slikker, M. 2006. Relaxed sequencing games have a nonempty core. *Naval Research Logistics* **53** 235–242.

Smith, W.E. 1956. Various optimizers for single-stage production. *Naval Research Logistics Quarterly* **3** 59–66.

Steiner, G., R. Zhang. 2009. Approximation algorithms for minimizing the total weighted number of late jobs with late deliveries in two-level supply chains. *Journal of Scheduling* **12** 565–574.

Thompson, S., M. Nunez, R. Garfinkel, M.D. Dean. 2009. Efficient short-term allocation and reallocation of patients to floors of a hospital during demand surges. *Operations Research* **57** 261–273.

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

Yang, B. 2007. Single machine rescheduling with new jobs arrivals and processing time compression. *International Journal of Advanced Manufacturing Technology* **34** 378–384.

Yu, G., M. Argüello, G. Song, S.M. McCowan, A. White. 2003. A new era for crew recovery at Continental Airlines. *Interfaces* **33** 5–22.

Yu, G., X. Qi. 2004. *Disruption Management: Framework, Models and Applications.* World Scientific, Singapore.

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

Appendices

Proof of Lemma 1

(a). Suppose there is an optimal schedule σ^*. First, consider the earlier partial schedule σ_A^* in an optimal schedule σ^*. If property (a) does not hold, then let j and i be the first processed pair of jobs (j, i), for which i precedes (completes earlier than) j in π^* but j immediately precedes (there is no other jobs processed or machine idle time in between) i in σ^*. We then generate a new schedule σ' by interchanging jobs j and i, where job i starts at time $S_j(\sigma^*)$ and job j starts at time $S_j(\sigma^*) + p_i$. If $C_j(\sigma') \geq C_j(\pi^*)$ (note that this indicates $C_i(\sigma') \geq C_i(\pi^*)$), then $\Delta_i(\pi^*, \sigma') < \Delta_i(\pi^*, \sigma^*)$ and $\Delta_j(\pi^*, \sigma') = C_j(\sigma') - C_j(\pi^*) = C_i(\sigma^*) - C_j(\pi^*) < C_i(\sigma^*) - C_i(\pi^*) = \Delta_i(\pi^*, \sigma^*)$, and therefore $\Delta_{\max}(\pi^*, \sigma') \leq \Delta_{\max}(\pi^*, \sigma^*)$, i.e., the maximum time disruption does not increase and schedule σ' remains feasible. Alternatively, consider the case where $C_j(\sigma') < C_j(\pi^*)$. If $C_i(\sigma') \leq C_i(\pi^*)$, then $\Delta_i(\pi^*, \sigma') \leq \Delta_j(\pi^*, \sigma') < \Delta_j(\pi^*, \sigma^*)$; if $C_i(\sigma') > C_i(\pi^*)$, then $\Delta_i(\pi^*, \sigma') < \Delta_i(\pi^*, \sigma^*)$ and $\Delta_j(\pi^*, \sigma') < \Delta_j(\pi^*, \sigma^*)$. i.e,, schedule σ' remains feasible as well. Hence we have $\Delta_{\max}(\pi^*, \sigma') \leq \Delta_{\max}(\pi^*, \sigma^*)$ under all conditions. Further, since π^* is obtained by the SWPT index rule, we have that the cost of σ' is no more than that of σ^*, and consequently σ' is an optimal schedule. A finite number of repetitions of this argument establishes part (a).

(b). The sequence of jobs in the latter schedule is established by identical interchange arguments provided in part (a). □

Proof of Lemma 2

Proof. Assume that there exists an optimal schedule σ^* with more than one single inserted idle time period in the earlier partial schedule σ_A^*. We then show that schedule σ^* can be adjusted to satisfy (a), (b) and (c), without increasing the total weighted completion time or the maximum time disruption.

According to Lemma 1, we may assume that jobs in σ_A^* of σ^* are processed in the same sequence as in π^*, i.e., in an SWPT index order. Let job i be the job immediately following the first inserted idle time period in σ_A^* of σ^*. Let $j_3 = \max\{j | C_j(\pi^*) - k \leq T_1\}$. Here, job j_3 is the last processed job in π^* that can be scheduled in σ_A^* of σ^* with a time disruption of no more than k time units. By our definition of i and j_3, we can see that there do not exist any other jobs besides jobs $i, i+1, \ldots, j_3$ that can be scheduled in σ_A^* after an inserted idle time period

without increasing the maximum time disruption. We further note that $S_i(\sigma^*) = S_i(\pi^*) - k$, for otherwise job i could be processed earlier to reduce the length of the idle time period it follows, without increasing the maximum time disruption or its scheduling cost. Note that there is no inserted idle time in π^*. Therefore, scheduling jobs $i, i+1, \ldots, j_3$ after the first idle time period consecutively without inserted idle time between any of them is feasible, where each job is scheduled exactly k time units earlier than in schedule π^*. Obviously, scheduling jobs $i, i+1, \ldots, j_3$ in such as way will neither increase the total weighted completion time or the maximum time disruption of schedule σ^*. This constructs an optimal schedule that satisfies (a), (b) and (c). □

Proof of Lemma 3

Proof. Suppose there is an optimal schedule σ^*. According to Lemma 1, we preceded to assume that jobs in σ_A^* of σ^* are processed in the same sequence as in π^*, i.e, in an SWPT index. Assume that job j immediately follows an inserted idle time period in σ_A^* of σ^*. By parts (a) and (b) of Lemma 2, we further assume that the time disruption of job j is k, which is also the maximum time disruption of schedule σ^*. By contradiction, next let us assume that job j has a start time no later than T_2 in π^*, i.e., $S_j(\pi^*) \leq T_2$.

By assumptions, we have that $S_j(\sigma^*) + k = S_j(\pi^*) \leq T_2$, i.e., $k \leq T_2 - S_j(\sigma^*)$. However, since no idle time period exists in π^*, the time interval $[0, S_j(\sigma^*)]$ is insufficient for the processing of all jobs that are scheduled from time 0 to time $S_j(\sigma^*)$ in schedule π^*. Note that jobs in σ_A^* of σ^* are processed in the same sequence as in π^*, and thus there must exist a job, denoted j', that starts before $S_j(\sigma^*)$ in π^* and is processed in σ_B^* of σ^*, i.e., starts at time T_2 or later in schedule σ^*. Therefore, the time disruption of job j' is $\Delta_{j'}(\pi^*, \sigma^*) > T_2 - S_j(\sigma^*) \geq k$. This contradicts the assumption that the maximum time disruption of schedule σ^* is exactly k time units. □

Proof of Lemma 4

Proof. Suppose there is an optimal schedule σ^*. According to Lemma 1, we may assume that jobs in the partial schedules σ_A^* and σ_B^* of an optimal schedule σ^* are processed in the same sequence as in π^*, i.e., SWPT order, each as early as possible. By contradiction, we assume that there exists an optimal schedule σ^* with at least one inserted idle time period in the later partial schedule σ_B^*.

Specifically, suppose that job i is scheduled in σ_B^* and is preceded immediately by an inserted idle time period. Note that all jobs processed earlier than job i in the initial optimal schedule π^* are still processed earlier than i in σ^*. Therefore, with the machine disruption, in schedule σ^* job i is processed no earlier than in π^*. Thus, the time disruption of job i is nondecreasing with its job completion time. Also, the weighted completion time of job i is also nondecreasing with its job completion time. Therefore, scheduling job i one time unit earlier, without changing the schedule of any other jobs, will result in a feasible schedule with total weighted completion time no less than that of σ^*, which contradicts the assumption that in the initial optimal schedule σ^*, each job is processed as early as possible in an SWPT index sequence. □

Proof of Lemma 5

Proof. We assume that an optimal schedule σ^* satisfies Lemmas 1 and 4. Under these assumptions, we can observe that no job in the later partial schedule σ_B^* of schedule σ^* can be completed earlier than in π^*. Suppose job j is processed the first in the later partial schedule σ_B^* of σ^* and is completed l time units later than in π^*. By contradiction, we preceded to assume a job j' processed later than job j in schedule σ^* is competed l' time units later than in schedule π^*, where l' is strictly larger than l.

By Lemma 1, we have that in both schedules π^* and σ^*, job j is processed earlier than job j'. Since there is no inserted idle time period in the later partial schedule σ_B^* of schedule σ^*, the condition $l' > l$ means that the total processing time of jobs between jobs j and j' in schedule σ^* is strictly larger than the total processing time of jobs between jobs j and j' in schedule π^*. This contradicts the assumption that jobs in the later partial schedule σ_B^* of schedule σ^* are processed in the same sequence as in π^*. □

Proof of Lemma 6

Proof. Suppose there is an optimal schedule σ^*. We consider the value of $\Delta_{\max}(\pi^*, \sigma^*)$, where schedule σ^* is an optimal schedule of problem $1|a, |\sum w_j C_j$ and satisfies Lemmas 1 and 2. We first consider jobs with $C_j(\sigma^*) > C_j(\pi^*)$. If $C_j(\sigma^*) \leq T_2$, then $\Delta_j \leq T_2$; if $C_j(\sigma^*) > T_2$, Lemma 1 indicates that all the jobs completed after time T_2 and before job j in schedule σ^* are processed before job j in schedule π^*, and thus $\Delta_j \leq T_2$. Alternatively, when $C_j(\sigma^*) \leq C_j(\pi^*)$, we have that $\Delta_j \leq P - p_{\min}$, since the maximum completion time of

schedule π^* is P. Thus, $\Delta_{\max}(\pi^*, \sigma^*) \leq \max\{T_2, P - p_{\min}\}$, and therefore, σ^* is an optimal schedule for problem $1|a, \Delta_{\max} \leq k| \sum w_j C_j$ with $k \geq \max\{T_2, P - p_{\min}\}$. □

Effects of D and k on Percentages of Rescheduling Cost and Cost Saving

First we consider how the percentage of rescheduling cost changes. First, it is evident that the percentage rescheduling cost increases with duration of the machine unavailable period. This is due to the assumption that the scheduling cost of each job proportionally increases with its completion time. Second, the percentage rescheduling cost decreases with the maximum allowable disruption k of the rescheduling process, but such an effect becomes less evident when the value of k is large, such as close to $\lfloor 4P/n \rfloor$. This observation indicates that to find an optimal schedule in rescheduling, it is not necessary to move jobs too far away, such as more than four times the average job processing time, from its position in the original schedule.

Second we discuss how the percentage of cost saving changes. First, the percentage cost saving slightly increases with the duration of the machine unavailable period. Second, the percentage cost saving increases with the maximum allowable disruption from the original schedule. The two relationships can be explained by the intuition that a larger value of D, or a larger value of k, would provide a larger space for Algorithm 1 to improve the heuristic schedule found by Algorithm 2. In addition, note that the effects of factor D on the percentage of cost saving are in the same pattern as on the percentage rescheduling cost, but are in a smaller magnitude (0.77% versus 4.74% on average). However, the effects of the maximum allowable time disruption k are different. This is because, a larger k provides more flexibility for rescheduling and thus reduces the extra rescheduling cost compared with the initial optimal schedule, while on the other hand, a larger k offers more flexibility for Algorithm 1 to save relative to Algorithm 2.