OPERATIONS RESEARCH
Vol. 58, No. 3, May–June 2010, pp. 746–755
ISSN 0030-364X | EISSN 1526-5463 | 10 | 5803 | 0746

DOI 10.1287/opre.1090.0751
© 2010 INFORMS

Rescheduling for Job Unavailability

Nicholas G. Hall
Fisher College of Business, The Ohio State University, Columbus, Ohio 43210, hall.33@osu.edu

Chris N. Potts
School of Mathematics, University of Southampton, Southampton SO17 1BJ, United Kingdom, c.n.potts@soton.ac.uk

This paper considers scheduling problems where the processing of a set of jobs has been scheduled (i.e., planned) to minimize a classical cost objective, under the assumption that the jobs are all available at the start of the planning horizon. Before processing starts, however, the availability of a subset of the jobs is delayed. Therefore, the decision maker needs to adjust the existing schedule to allow for the initial unavailability of those jobs, but without causing excessive disruption to the schedule and expensive resource reallocations. The limit on allowable disruption is measured by the maximum time disruption to any job, between the original and adjusted schedules. For the classical sum of weighted completion times scheduling objective, we provide a computationally efficient optimal algorithm and an intractability proof showing that such an algorithm is the best possible type of result. Also, we provide a linear time approximate solution procedure, show that its worst-case performance ratio is a small constant, and demonstrate computationally that its average performance is very close to optimal. Finally, we provide a fully polynomial time approximation scheme. We also summarize analogous results for three other classical scheduling objectives. Our work is among the first to develop optimal algorithms, heuristics with guaranteed performance bounds, and approximation schemes, for rescheduling problems.

Subject classifications: production/scheduling: sequencing, deterministic; single machine; manufacturing, performance/productivity.
Area of review: Optimization.
History: Received September 2005; revisions received August 2006, August 2007, August 2008; accepted September 2008. Published online in *Articles in Advance* December 8, 2009.

1. Introduction

A natural consequence of the frequent, unexpected *disruptions* that occur in manufacturing practice is the need for *rescheduling*. Among the most common types of disruption are: changes in release dates, the arrival of new orders, order cancellations, changes in order priority, processing delays, and machine breakdowns. Rescheduling involves adjusting a previously planned schedule to account for a disruption. This involves a trade-off between finding a cost-efficient new schedule and avoiding excessive changes to the previously planned schedule.

We consider the issue of how to reschedule jobs in a manufacturing environment that experiences a disruption. We assume that a set of jobs has been scheduled optimally to minimize some classical cost objective. However, the processing of many of those jobs has not begun. This situation arises wherever schedules are planned in advance of their start date, which in practice typically occurs hours or even days earlier. Based on the planned schedule, resource allocations and delivery dates to customers have been planned. Then, either before or during the execution of the schedule, the dates at which some of the jobs are available to start processing, known in the scheduling literature as *release dates*, change. This most frequently occurs as a result of poor supplier performance.

In recent years, rescheduling has attracted a considerable amount of research attention. This interest appears to be motivated by both important practical issues and interesting research problems. The need for rescheduling in response to unexpected changes that take place in production environments is commonplace in modern flexible decision-making and manufacturing systems (Vieira et al. 2003). We discuss several specific examples of practical rescheduling problems from industry. Bean et al. (1991) consider an automobile industry application for which they develop a match-up scheduling approach that compensates for subsequent disruptions. Zweben et al. (1993) describe the GERRY scheduling and rescheduling system for ground processing of the space shuttle. Clausen et al. (2001) describe a shipyard application that involves the management of large steel plates; when a disruption changes the order in which the plates are needed, then rescheduling is required. Yu et al. (2003), in their work that received the 2002 Edelman Award of INFORMS, describe an optimization-based approach for rescheduling aircraft operations to compensate for disruptions.

Extensive reviews of the rescheduling literature are provided by Davenport and Beck (2000), Vieira et al. (2003), Aytug et al. (2005), and Herroelen and Leus (2005). Therefore, we review only the works that are directly related to this paper. Wu et al. (1992) consider the problem of rescheduling a job shop after a disruption to minimize the makespan and the total difference between the start times of operations in the original and new schedules. Unal et al. (1997) insert new jobs into an original schedule to minimize the total weighted completion time or makespan of the new jobs, without incurring additional setups or causing jobs to become late. They provide efficient algorithms, intractability results, and heuristics with guaranteed performance bounds. Hall and Potts (2004) consider scheduling problems where a new set of jobs arrives and creates a disruption to a planned schedule of original jobs. Hall et al. (2007) consider multiple arrivals of new jobs.

This paper investigates the trade-off between the scheduling cost of the jobs and the extent of the disruption that arises from the unavailability of some of the jobs, to the previously planned schedule. Our scheduling cost is the sum of weighted completion times of the jobs, although we also summarize analogous results for the maximum lateness, the makespan, and the sum of completion times. In many situations—for example, those where customer goodwill is reduced by a disruption—the cost of the disruption may be hard to estimate. Therefore, we model the amount of disruption as a constraint rather than as a cost. The paper provides optimal rescheduling methods that require very little computation time, in most cases only a few seconds, even for large problems. Approximation methods that run even faster, in linear time, and have small worst-case bounds and are very close to optimal average performance, are also described. Because those methods run so quickly, they can easily be used for rescheduling in even the most rapidly changing manufacturing environment.

This paper is organized as follows. In §2, we introduce our notation and define our modeling environment. Section 3 presents an optimization algorithm and an intractability result for the classical scheduling objective of minimizing the total weighted completion time. Section 4 presents an approximation algorithm, and analyzes its worst-case and average performance. Section 5 describes a fully polynomial time approximation scheme. In each of §§3–5, we also summarize analogous results for three other classical scheduling objectives. Finally, §6 contains a summary of our results and suggestions for future research.

2. Model Definitions

In this section, we describe our notation and problem classification scheme, and we formally define our modeling environment. Let $J = \{1, \ldots, n\}$ denote a set of jobs to be processed nonpreemptively on a single machine. Let $p_j > 0$ and $w_j \geqslant 0$ denote the processing time and weight of job j, respectively, for $j = 1, \ldots, n$. We assume throughout that all p_j and w_j are known integers. Let $p_{\max} = \max_{j\in J}\{p_j\}$, $P = \sum_{j=1}^{n} p_j$, and $W = \sum_{j=1}^{n} w_j$. We assume that the jobs of J have been scheduled optimally to minimize some classical objective and that π^* is the resulting optimal schedule with no idle time between the jobs.

Let $R \subseteq J$ denote a subset of the jobs that, as a result of a disruption, will not be available for processing until some integer time $r > 0$. Possible extensions to allow multiple release dates for the jobs of R are discussed in §6. In the original problem all jobs are available for processing at time zero, and in the disrupted problem the jobs of $J\backslash R$ are also available for processing at time zero. We assume that this information becomes available after π^* has been determined, but before processing begins. If the new information becomes available after the start of processing, then the processed jobs of J are removed from the problem, any partly processed jobs of J are processed to completion, and J is updated accordingly. If $R = J$, then the problem is equivalent to a processing delay or machine breakdown that prevents the processing of any jobs during the interval $[0, r]$.

For any schedule σ of the jobs of J, we define the following variables for $j \in J$:

$C_j(\sigma) =$ the time at which job j is completed;

$\Delta_j(\sigma) = |C_j(\sigma) - C_j(\pi^*)|$, the *time disruption* of job j.

When there is no ambiguity, we simplify $C_j(\sigma)$ and $\Delta_j(\sigma)$ to C_j and Δ_j, respectively. Let σ^* denote an optimal schedule.

The standard classification scheme for scheduling problems (Graham et al. 1979) is $\alpha|\beta|\gamma$, where α indicates the scheduling environment, β describes the job characteristics or restrictive requirements, and γ defines the objective function to be minimized. We consider single machine problems, thus implying that $\alpha = 1$. Under β, we write r_R to denote that all the jobs in the set R have release date r. Also under β, we use $\Delta_{\max} \leqslant k$ to indicate that $\max_{j\in J}\{\Delta_j\} \leqslant k$, where k is a positive integer. That is, no job can have a time disruption that exceeds k. This constraint ensures that all customers receive a reasonable level of service. Under β, we also write *pmtn* when preemption is allowed in order to obtain a lower bound. Our results focus mainly on the $\sum w_j C_j$ objective, or more formally $\gamma = \sum_{j\in J} w_j C_j$, the total weighted completion time of the jobs. However, we also summarize some results for $\gamma \in \{C_{\max}, L_{\max}, \sum C_j\}$, where $C_{\max} = \max_{j\in J}\{C_j\}$ is the makespan of the jobs, $L_{\max} = \max_{j\in J}\{C_j - d_j\}$ is the maximum lateness of the jobs, where d_j represents a given due date for job j, and $\sum C_j = \sum_{j\in J} C_j$ is the total completion time of the jobs.

The choice of π^* for the total weighted completion time objective is defined by a shortest weighted processing time (SWPT) sequence in which the jobs appear in nondecreasing order of p_j/w_j (Smith 1956). We index the jobs so that π^* is defined by the sequence $(1, \ldots, n)$. For notational convenience, we write $\pi^* = (1, \ldots, n)$. The sequences that define π^* for any of the objective functions considered in this paper are constructed by *priority index rules*, indicating that given an arbitrary sequence where an adjacent pair of jobs appears in the reverse order to that in π^*, an interchange of this pair of jobs leads to a schedule in which the value of γ is either smaller or the same.

3. Optimization Algorithms and Intractability

3.1. General Properties

The constraint $\Delta_{\max} \leqslant k$ implies that, in a feasible schedule σ, $C_j(\sigma) \geqslant C_j(\pi^*) - k$ and $C_j(\sigma) \leqslant C_j(\pi^*) + k$, for $j = 1, \ldots, n$. Accordingly, each job j has an *implied release date* $\bar{r}_j = C_j(\pi^*) - p_j - k$ and an *implied deadline* $\bar{d}_j = C_j(\pi^*) + k$. We refer to the partial schedule of jobs that start processing before time r as the *earlier schedule*; jobs that start processing at time r or later form the *later schedule*. All jobs of R are contained in the later schedule. Also, let $h = \min\{i \mid i \in R\}$.

We first consider a special case of problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \gamma$ that is easy to solve.

Lemma 1. *In problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \gamma$, if $\sum_{i=1}^{h-1} p_i \geqslant r$, then π^* is optimal.*

Proof. Because π^* provides an optimal schedule for the unconstrained problem $1 \| \gamma$, and under the stated condition that the schedule is also feasible for the constrained problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \gamma$, the result follows immediately. □

In view of Lemma 1, we henceforth assume that

$$\sum_{i=1}^{h-1} p_i < r. \tag{1}$$

Moreover, if $r > \bar{d}_h - p_h = C_h(\pi^*) - p_h + k$, then job h cannot be scheduled within the allowable time disruption, and the problem is infeasible. Therefore, we assume that

$$r \leqslant C_h(\pi^*) - p_h + k. \tag{2}$$

We now establish that inequality (2) is sufficient to ensure feasibility.

Lemma 2. *Under condition* (2), *problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \gamma$ is feasible.*

Proof. We construct a schedule σ in which the earlier schedule comprises jobs $1, \ldots, h-1$ that are scheduled in the same time intervals as in π^*, and the later schedule comprises jobs $h, \ldots, n$ that are scheduled in this order, starting at time r, with no idle time between the jobs. Clearly, the earlier schedule is feasible. For any job j in the later schedule, $C_j(\sigma) = r + \sum_{i=h}^{j} p_i \leqslant C_h(\pi^*) - p_h + k + \sum_{i=h}^{j} p_i = C_j(\pi^*) + k = \bar{d}_j$, where the inequality follows from (2), and also $C_j(\sigma) \geqslant C_j(\pi^*) \geqslant \bar{r}_j$. Therefore, σ is feasible. □

From the assumption that (2) holds and from Lemma 2, we restrict our attention to problem instances that are feasible.

Our next result establishes the order of jobs in the earlier and later schedules.

Lemma 3. *There exists an optimal schedule σ^* for problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \gamma$ in which*:

(a) *the jobs in the earlier schedule are sequenced in the same order as in π^**;

(b) *the jobs in the later schedule are sequenced in the same order as in π^*.*

Proof. (a) Consider first the earlier schedule in σ^*. If property (a) is not satisfied, let j and i be the first pair of jobs ($j, i \in J \backslash R$) for which i precedes j in π^* but j immediately precedes i in σ^*. Because $C_i(\pi^*) < C_j(\pi^*)$, we have $\bar{r}_i < \bar{r}_j$ and $\bar{d}_i < \bar{d}_j$. Interchange jobs j and i to obtain a new schedule σ', where jobs i and j occupy the same time interval as in σ^*, with job i starting at time $C_j(\sigma^*) - p_j$ and job j completing at time $C_i(\sigma^*)$. Because $C_i(\sigma') - p_i = C_j(\sigma^*) - p_j \geqslant \bar{r}_j > \bar{r}_i$ and $C_j(\sigma') = C_i(\sigma^*) \leqslant \bar{d}_i < \bar{d}_j$, it follows that σ' is feasible. Moreover, because π^* is obtained by a priority index rule, we have $\gamma(\sigma') \leqslant \gamma(\sigma^*)$. Therefore, σ' is also an optimal schedule. A finite number of repetitions of this argument establishes part (a).

(b) An identical interchange argument to that used in part (a) establishes the sequence of the jobs in the later schedule. □

We now establish a result that provides more specific properties about the structure of the earlier and later schedules.

Lemma 4. *There exists an optimal schedule σ^* for problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \gamma$, where all jobs are started as early as possible, in which*:

(a) *$C_j(\sigma^*) \leqslant C_j(\pi^*)$ for each job j in the earlier schedule*;

(b) *jobs $1, \ldots, h-1$ appear in index order at the start of the schedule*;

(c) *job h is the first job in the later schedule*;

(d) *$C_j(\sigma^*) \geqslant C_j(\pi^*)$ and $C_j(\sigma^*) - C_j(\pi^*) \leqslant C_h(\sigma^*) - C_h(\pi^*)$, for each job j in the later schedule.*

Proof. We consider each part separately.

(a) Every job that precedes any job j in the earlier schedule in σ^* also precedes job j in π^*, as a consequence of part (a) of Lemma 3. Because the jobs in σ^* start as early as possible, the desired result follows.

(b) If each of jobs $1, \ldots, h-1$ appears in the earlier schedule in σ^*, then the result follows from part (a) of Lemma 3. Otherwise, let $i \leqslant h-1$ denote the job with the smallest index that appears in the later schedule in σ^*, where $i \in J \backslash R$ from the definition of h. Part (b) of Lemma 3 establishes that job i appears first in the later schedule. Moreover, part (a) of Lemma 3 establishes that jobs $1, \ldots, i-1$ appear at the start of the earlier schedule and are scheduled in the same time intervals as in π^*, with no idle time between these jobs. Job i is immediately preceded by some job j in σ^*, where j is in the earlier schedule. We note that there is no idle time between jobs j and i because otherwise job i would be scheduled earlier, and that $j > i$. The interchange argument used in part (a) of the proof of Lemma 3 shows that jobs j and i can be interchanged, thereby yielding an alternative optimal schedule. Moreover, a finite number of interchanges of this type yields an optimal schedule in which jobs $1, \ldots, h-1$ are sequenced in index order at the start of the earlier schedule.

(c) From the definition of h, we deduce from part (b) of Lemma 3 that job h is the first job in the later schedule.

(d) The sequencing results of Lemma 3 show that each job j in the later schedule in σ^* is preceded by jobs $1, \ldots, j-1$, and recall that only these jobs precede job j in π^*. Therefore, $C_j(\sigma^*) \geqslant C_j(\pi^*)$ as required, and this implies that σ^* has no idle time between jobs in the later schedule. Moreover, $C_j(\sigma^*) - C_h(\sigma^*) \leqslant C_j(\pi^*) - C_h(\pi^*)$, which implies that $C_j(\sigma^*) - C_j(\pi^*) \leqslant C_h(\sigma^*) - C_h(\pi^*)$. $\square$

Part (a) of Lemma 4 shows that no job in the earlier schedule in σ^* can complete later than it does in π^*, and therefore the implied deadlines of jobs in the earlier schedule are redundant for schedules constructed in accordance with Lemmas 3 and 4. Similarly, the implied release dates are redundant for jobs in the later schedule, as a result of part (d). Moreover, part (d) also shows that the feasibility of the later schedule in σ depends only on the feasibility of its first job h, without the need to check that other jobs satisfy their implied deadlines.

We now present another special case that is based on a partitioning of the problem.

Lemma 5. *In problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \gamma$, if*

$$\max_{j \in J \setminus R} C_j(\pi^*) \leqslant r,$$

then an optimal solution schedules the jobs of $J \setminus R$ in π^* order as early as possible within the interval $[0, \max_{j \in J \setminus R} C_j(\pi^*)]$ and schedules the jobs of R in π^* order as early as possible, starting at time r.

Proof. Under the condition of the lemma, optimal schedules can be found for the jobs of $J \setminus R$ (in the earlier schedule) and of R (in the later schedule), independently of each other. Parts (a) and (b), respectively, of Lemma 3 define the processing sequences for these jobs. $\square$

Lemma 5 specifies conditions under which a partitioning approach solves the problem. To avoid such cases, we henceforth restrict our attention to instances in which

$$r < \max_{j \in J \setminus R} C_j(\pi^*). \tag{3}$$

From the above results, we may restrict our search for an optimal schedule for which:

(a) the earlier schedule starts with jobs $1, \ldots, h-1$ and sequences its jobs in π^* order, while respecting the implied release date constraints;

(b) the later schedule starts with job h and sequences its jobs in π^* order without idle time between jobs, with feasibility ensured if the implied deadline constraint for job h is satisfied.

3.2. Total Weighted Completion Time

We present a dynamic programming algorithm problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \sum w_j C_j$. The algorithm uses the following *preprocessing step*: Find h, the start job of the later schedule, and compute the implied release dates $\bar{r}_j = C_j(\pi^*) - p_j - k$ for $j \in J \setminus R$, and the implied deadline $\bar{d}_h = C_h(\pi^*) + k$ of job h. Note that π^* indexes the jobs in SWPT order.

Algorithm TWCOA

Value Function. $f(j, t_1, t_2, w) =$ minimum total weighted completion time for a schedule of jobs $1, \ldots, j$, where jobs of the earlier schedule occupy the interval $[0, t_1]$, jobs of the later schedule are currently scheduled in the interval $[0, t_2]$ but will subsequently be appended to the earlier schedule, and jobs of the later schedule have a total weight of w.

Boundary Condition.

$$f(h, t_0, p_h, w_h) = \sum_{i=1}^{h-1} w_i \sum_{j=1}^{i} p_j + w_h p_h, \quad \text{where } t_0 = \sum_{i=1}^{h-1} p_i.$$

Optimal Solution Value.

$$\min_{t_0 \leqslant t_1 \leqslant T,\, p_h \leqslant t_2 \leqslant P - t_0,\, w_h \leqslant w \leqslant W - w_0} \{f(n, t_1, t_2, w) + w \max\{t_1, r\}\},$$

where $T = \min\{P, r + p_{\max} - 1, \bar{d}_h - p_h\}$ and $w_0 = \sum_{i=1}^{h-1} w_i$.

Recurrence Relation.

$$f(j, t_1, t_2, w) = \min \begin{cases} f(j-1, t_1, t_2 - p_j, w - w_j) + w_j t_2 \\ f(j-1, t_1 - p_j, t_2, w) + w_j t_1, \\ \qquad\qquad \text{if } j \in J \setminus R,\ \bar{r}_j + p_j < t_1 \\ \displaystyle\min_{t_0 \leqslant t_1' \leqslant t_1 - p_j} \{f(j-1, t_1', t_2, w) + w_j t_1\}, \\ \qquad\qquad \text{if } j \in J \setminus R,\ \bar{r}_j + p_j = t_1 \end{cases}$$

for $j = h+1, \ldots, n$, $t_1 = t_0, \ldots, T$, $t_2 = p_h, \ldots, P - t_0$, $w = w_h, \ldots, W - w_0$.

In Algorithm TWCOA, we initialize the earlier and later schedule with jobs $1, \ldots, h-1$ occupying the earlier schedule starting at time zero, and job h occupying the later schedule starting at time zero. The first equation in the recurrence relation places job j in the later schedule to complete at time t_2. The second and third equations place job j in the earlier schedule to complete at time t_1. Job j starts after its implied release date $\bar{r}_j$ in the second equation and therefore is not preceded by idle time, but starts at its implied release date in the third equation, thereby allowing the possibility of idle time immediately before the start of its processing. In the optimal solution value step, the earlier and later schedules are concatenated so that the later schedule starts at time $\max\{t_1, r\}$.

Theorem 1. *Algorithm TWCOA finds an optimal schedule for problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \sum w_j C_j$ in $O(nPTW)$ time.*

Proof. Because Algorithm TWCOA uses the structural properties that are justified in Lemmas 3 and 4, and compares the cost of all possible state transitions, it generates an optimal schedule. From the inequalities $j \leqslant n$, $t_1 \leqslant T$, $t_2 \leqslant P$, and $w \leqslant W$, the number of possible values for the state variables is $O(nPTW)$. The first and second equations in the recurrence relation require constant time. The third equation requires $O(T)$ time, but is computed for only one value of t_1 for each value of j. Therefore, the overall time complexity of Algorithm TWCOA is $O(nPTW)$. $\square$

The following negative result for problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \sum w_j C_j$ shows that the pseudopolynomial time algorithm in Theorem 1 provides the best type of result possible, unless $P = NP$.

THEOREM 2. *The recognition version of problem* $1 \mid r_R, \Delta_{\max} \leqslant k \mid \sum w_j C_j$ *is binary NP-complete, even for the case of unit weights.*

PROOF. For k sufficiently large, the constraint becomes redundant. Because Rinnooy Kan (1976) shows that the recognition version of the classical problem $1|r_j|\sum C_j$ is binary *NP*-complete for the case of a single nonzero release date, the result follows. $\square$

3.3. Other Objectives

The online companion, which is available as part of the online version that can be found at http://or.journals.informs.org/, provides a detailed analysis of optimal algorithms that are similar to TWCOA for problems $1 \mid r_R, \Delta_{\max} \leqslant k \mid \sum L_{\max}$, and $1 \mid r_R, \Delta_{\max} \leqslant k \mid C_{\max}$. We now provide an overview of that analysis.

For the maximum lateness problem, the algorithm with a time complexity of $O(nPT)$ is simpler than TWCOA in that the state variable w is eliminated. For the makespan problem, a further simplification is possible, with a single state variable that represents idle time replacing both t_1 and t_2. This reduces the time complexity to $O(nT)$. In both cases, the three equations in the recurrence relation are similar to those in Algorithm TWCOA. The recognition versions of both problems are binary NP-complete.

For problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \sum C_j$, the application of Algorithm TWCOA requires $O(n^2PT)$ time. It follows from Theorem 2 that the recognition version of this problem is binary NP-complete.

4. Approximation Algorithms

4.1. Lower Bound

For problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \gamma$, we propose the following generic approximation algorithm (GAA). The jobs are considered in π^* order, and those in $J \backslash R$ are scheduled as early as possible in the earlier schedule, provided that this allows the later schedule to start at time r. The remaining jobs are scheduled in π^* order in the later schedule. A formal description now follows.

Algorithm GAA

Initialization. Schedule jobs $1, \ldots, h-1$ in index order in the interval $[0, \sum_{i=1}^{h-1} p_i]$, and schedule job h in the interval $[r, r+p_h]$.

Scheduling. For jobs $h+1, \ldots, n$, schedule each at the earliest feasible time, subject to not delaying a previously scheduled job.

Output. Output the resulting schedule, σ^H, and its cost, z^H.

The requirement not to move a previously scheduled job later in the scheduling step of Algorithm GAA is motivated by the observation that doing so may result in infeasibility or an increase in the cost of the schedule. Algorithm GAA requires $O(n)$ time, and the feasibility of the resulting schedule is guaranteed by Property (b) in the discussion following Lemma 5.

In our analysis of the performance of Algorithm GAA, we use z^* to denote the optimal cost for a given problem instance. We derive worst-case performance bounds for Algorithm GAA by establishing an upper bound on the value of the ratio z^H/z^* over all problem instances.

To provide a lower bound on the value of z^* for the total weighted completion-time objective function, we consider a relaxation of problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \sum w_j C_j$. Because the recognition version of problem $1 \mid r_j, pmtn \mid \sum w_j C_j$ is unary *NP*-complete (Labetoulle et al. 1984), we consider a *job-splitting relaxation* (Belouadah et al. 1992), where a job may be split into pieces and a cost is incurred on completion of each piece. We propose the following algorithm to solve this relaxation.

Algorithm XOA

Initialization. Schedule jobs $1, \ldots, h-1$ in index order in the interval $[0, \sum_{i=1}^{h-1} p_i]$, and schedule job h in the interval $[r, r+p_h]$.

Scheduling. For jobs $h+1, \ldots, n$, schedule each to start at the earliest feasible time, subject to not delaying any previously scheduled job. If some job j is scheduled to start at time t, where $r - p_j < t < r$, then job j is split into two pieces j_1 and j_2, with j_1 occupying the time interval $[t, r]$, where $p_{j_1} = r - t$, $w_{j_1} = p_{j_1} w_j / p_j$, $p_{j_2} = p_j - p_{j_1}$, and $w_{j_2} = p_{j_2} w_j / p_j$. The pieces j_1 and j_2 are regarded as separate jobs, and after job j has been replaced by j_2, all remaining jobs are then processed in π^* order in the later schedule.

Output. Output the resulting schedule, σ^X, and its cost, z^X, where the contribution $w_{j_1} p_{j_2}$ is added to the cost when job j is split in the scheduling step.

Algorithm XOA creates at most one split, which occurs at time r. All subsequent jobs are then placed in the later schedule. We next establish that Algorithm XOA provides a lower bound on z^*.

THEOREM 3. *For problem* $1 \mid r_R, \Delta_{\max} \leqslant k \mid \sum w_j C_j$, *Algorithm XOA finds a lower bound* $z^X \leqslant z^*$.

PROOF. The result is proved by establishing the optimality of Algorithm XOA for the relaxed problem. Let σ denote an optimal schedule for the corresponding relaxed scheduling problem. Let the set of jobs or job pieces that complete processing up to time r be denoted E, and the set of jobs or job pieces that start processing after time r be denoted L. From the initialization and scheduling steps, Algorithm XOA schedules the lowest-indexed available jobs up to time r. Now assume that the jobs of E are not the lowest indexed available jobs in schedule σ. Then there exists a job $i \in (J\backslash R) \cap L$ or a job piece $l_i \in (J\backslash R) \cap L$ of job i in the later schedule in σ, where this job or job piece is chosen so that i is the lowest possible index, and a job $j \in E$ or a job piece $e_j \in E$ of job j, where $j > i$.

We now show how to construct an alternative optimal schedule, σ', where we allow any splitting of the jobs. Jobs i and j are removed from σ and the empty time intervals are filled, first scheduling all processing of job i as early as possible, and then filling the remaining intervals with processing of job j. Because $j > i$ implies that $\bar{r}_i < \bar{r}_j$ and $\bar{d}_i < \bar{d}_j$, this transformation results in a feasible schedule.

We consider the resulting change in cost. The SWPT indexing implies that $p_i/w_i \leqslant p_j/w_j$. For this analysis, the job-splitting principle allows us to regard the transformation from σ to σ' as comprising a series of interchanges of pieces of jobs i and j, where each piece has a unit processing time. Specifically, each interchange involves a unit piece of job j with weight w_j/p_j and a piece of job i with weight w_i/p_i, where the piece of job j is scheduled earlier. Because $i < j$ implies $w_j/p_j \leqslant w_i/p_i$, each interchange does not increase the value of the objective function, and therefore $\sum w_g C_g(\sigma') \leqslant \sum w_g C_g(\sigma)$.

Thus, σ' is optimal. A finite number of repetitions of the above interchange results in the same subsets of jobs in the earlier and later schedules, as obtained in the scheduling step of Algorithm XOA. Arguments similar to those in Lemma 3 show that an optimal sequence of jobs is obtained from the π^* order, as implemented in the initialization and scheduling steps of Algorithm XOA. □

4.2. Total Weighted Completion Time

We now consider the worst-case performance of Algorithm GAA for problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \sum w_j C_j$.

Theorem 4. *For problem* $1 \mid r_R, \Delta_{\max} \leqslant k \mid \sum w_j C_j$, *Algorithm GAA has a worst-case bound of* 2, *and no better performance bound exists for Algorithm GAA.*

Proof. If no job is split in schedule σ^X, then $\sigma^H = \sigma^X$, which implies that σ^H is optimal. Otherwise, some job j is split into two pieces j_1 and j_2 in σ^X, where piece j_1 is scheduled in the interval $[r - p_{j_1}, r]$. Here, let t denote the time from the completion of processing of piece j_1 to the start of processing of piece j_2 in σ^X. Also, let W^X denote the total weight of the jobs that follow piece j_2 in σ^X.

The main idea of the proof is to compare the schedules σ^H and σ^X. Jobs $1, \ldots, j-1$ are scheduled identically in the two schedules, so that

$$\sum_{i=1}^{j-1} w_i C_i(\sigma^H) = \sum_{i=1}^{j-1} w_i C_i(\sigma^X). \tag{4}$$

In schedule σ^X, piece j_1 is scheduled in the interval $[r - p_{j_1}, r]$, and piece j_2 is scheduled in the interval $[r+t, r+t+p_{j_2}]$. Thus, pieces j_1 and j_2 contribute $w_{j_1} r + w_{j_2}(r+t+p_{j_2}) + w_{j_1} p_{j_2}$ to the objective function, which includes the contribution added in the output step of Algorithm XOA. However, schedule σ^H processes job j in the interval $[r+t, r+t+p_j]$, which provides an objective function contribution of $w_j(r+t+p_j)$. Because $w_{j_2} p_{j_1} = w_{j_1} p_{j_2}$, $p_j = p_{j_1} + p_{j_2}$, and $w_j = w_{j_1} + w_{j_2}$, we have

$$\begin{aligned} &w_{j_1} C_{j_1}(\sigma^X) + w_{j_2} C_{j_2}(\sigma^X) + w_{j_1} p_{j_2} \\ &\quad = w_j r + w_{j_2} t + w_j p_{j_2} = w_j C_j(\sigma^H) - w_{j_1} t - w_j p_{j_1}, \end{aligned}$$

which implies that

$$\begin{aligned} w_j C_j(\sigma^H) = {}& (w_{j_1} C_{j_1}(\sigma^X) + w_{j_2} C_{j_2}(\sigma^X) + w_{j_1} p_{j_2}) \\ & + w_{j_1} t + w_j p_{j_1}. \end{aligned} \tag{5}$$

Finally, we consider the jobs $j+1, \ldots, n$ scheduled after piece j_2 in schedule σ^X. In schedule σ^H, some of these jobs may be scheduled in the interval $[r - p_{j_1}, r]$. However, the construction of the schedules ensures no job starts processing more than p_{j_1} units later in σ^H than in σ^X. Therefore,

$$\sum_{i=j+1}^{n} w_i C_i(\sigma^H) \leqslant \sum_{i=j+1}^{n} w_i C_i(\sigma^X) + W^X p_{j_1}. \tag{6}$$

Combining (4), (5), and (6), we obtain

$$\begin{aligned} z^H - z^* \leqslant z^H - z^X &\leqslant w_{j_1} t + w_j p_{j_1} + W^X p_{j_1} \\ &\leqslant w_j t + w_j p_j + W^X p_j. \end{aligned} \tag{7}$$

To complete our analysis, we establish a lower bound on z^* by considering job j and all jobs of the later schedule in σ^X excluding the piece j_2. Under this relaxation, these jobs are processed in SWPT order, starting from time zero, so that job j is processed in the interval $[t, t+p_j]$. We further relax the problem by setting the weights of jobs scheduled before job j to zero, and setting the processing times of jobs scheduled after job j to zero. Thus, we obtain

$$z^* \geqslant w_j(t+p_j) + W^X(t+p_j). \tag{8}$$

Combining inequalities (7) and (8) yields the desired inequality $z^H \leqslant 2z^*$.

The following instance of problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \sum w_j C_j$ shows that no better bound exists. The instance is defined by: $n = 3$; $p_1 = 1, p_2 = 1, p_3 = r$; $w_1 = 1$, $w_2 = 1$, $w_3 = r$; $\pi^* = (1, 2, 3)$; $R = \{1\}$; and $k = 2r$. In schedule σ^H, jobs 1, 2, and 3 are scheduled in the intervals $[r, r+1]$, $[0, 1]$, and $[r+1, 2r+1]$, respectively; thus, $z^H = (r+1) + 1 + r(2r+1) = 2r^2 + 2r + 2$. In schedule σ^*, jobs 1, 2, and 3 are scheduled in the intervals $[r, r+1]$, $[r+1, r+2]$, and $[0, r]$, respectively; thus, $z^H = (r+1) + (r+2) + r^2 = r^2 + 2r + 3$. Therefore,

$$\lim_{r\to\infty} z^H/z^* = \lim_{r\to\infty} (2 - (2r+4)/(r^2+2r+3)) = 2,$$

which shows that no better bound exists for Algorithm GAA. □

4.3. Other Objectives

The online companion provides a detailed analysis of the worst-case performance of Algorithm GAA for the other three scheduling objectives. We now provide an overview of that analysis.

For the maximum lateness objective, we make the standard assumption that all due dates are nonpositive; otherwise, the worst-case performance bound is not well defined. For both the maximum lateness and makespan problems, we establish a worst-case bound of 2. Our analysis shows that the difference in completion time between the σ^H and π^* schedules for any job j in the later schedule is bounded by the amount of idle time in the schedule. The bound follows because this idle time cannot exceed r, and $r \leqslant P \leqslant z^*$.

For the total completion time objective, a worst case bound of 5/3 is established by comparing σ^H with an optimal preemptive schedule. The result is proved by showing that the difference in the total completion time for these two schedules is bounded by $2z^*/3$.

4.4. Computational Testing

We test Algorithm GAA on 5,400 randomly generated problem instances. These instances are generated according to the four principles and eight properties required by a random generation scheme (Hall and Posner 2001). In particular, because our preliminary testing revealed that the value of k does not significantly affect the performance of Algorithm GAA, the parsimony property suggests that a midrange value of $k = r + 0.25(P - r)$ should be used throughout our main experiment. Also, we satisfy the variety property by performing experiments with 100, 300, and 1,000 jobs, where 10%, 20%, or 50% of the jobs have their start delayed until time r or later, i.e., $|R| = 0.1n$, $|R| = 0.2n$, or $|R| = 0.5n$. We randomly generate $p_j \sim U[a, \ldots, b]$ and $d_j \sim U[a, \ldots, (a+b)n/3]$, where $a = 1$ or $a = 81$ and $b = 100$, and $w_j \sim U[1, \ldots, 10]$, for $j = 1, \ldots, n$. Having computed $P = \sum_{j=1}^n p_j$, we generate instances with $r = 0.1P$, $r = 0.2P$, and $r = 0.5P$. For each value of n, $|R|$, a, and r, 100 random instances are generated, thus providing a total of 5,400 instances. We consider only problem instances that satisfy the feasibility condition (2) and the nontriviality conditions given by (1) and (3) through removing infeasible or trivial instances. The algorithms are coded in the C++ language, compiled using the Microsoft Visual C++ compiler, and run on a Pentium III 500 MHz personal computer with 128 MB of memory. The mean computation time for an instance with 1,000 jobs is approximately 0.11 seconds for both Algorithms GAA and XOA, where Algorithm XOA is modified in a straightforward way for the maximum lateness, makespan, and total completion time objective functions by preempting jobs rather than splitting them.

Our computational results are shown in Tables 1 and 2. Columns 2 through 9 list relative percentage errors of the solution cost, z^H, obtained from Algorithm GAA, compared to the lower bound, z^X, on the optimal cost, i.e., $100(z^H - z^X)/z^X$. The second column lists the means, TWC, of the relative percentage error over 1,800 randomly generated problem instances in each row for the $\sum w_j C_j$ objective, whereas the third column lists the largest relative percentage errors, TWC^+. Similar results appear for $C_{\max}$ as C and C^+ in Columns 4 and 5; for $L_{\max}$ as L and L^+ in Columns 6 and 7; and for $\sum C_j$ as TC and TC^+ in Columns 8 and 9.

Table 1. Effect of n on performance of Algorithm GAA.

n	TWC	TWC^+	C	C^+	L	L^+	TC	TC^+
100	0.75	1.97	0.55	1.09	1.15	3.23	0.71	1.45
300	0.28	1.64	0.18	0.36	0.37	0.93	0.22	0.48
1,000	0.08	0.19	0.05	0.11	0.10	0.26	0.07	0.14

Table 2. Effect of r/P on performance of Algorithm GAA.

r/P	TWC	TWC^+	C	C^+	L	L^+	TC	TC^+
0.10	0.48	1.26	0.19	0.41	0.79	2.11	0.36	0.78
0.20	0.46	1.12	0.25	0.49	0.81	2.24	0.38	0.77
0.50	0.17	0.41	0.33	0.66	0.01	0.08	0.25	0.53

The results in Tables 1 and 2 indicate that Algorithm GAA is extremely effective at finding near optimal solutions. The mean relative error between the cost of the solution from Algorithm GAA and the lower bound from Algorithm XOA is 0.37% for the $\sum w_j C_j$ objective over the 5,400 problem instances tested. Similar results are achieved for the three other objectives. The largest relative errors of the solutions found by Algorithm GAA occur with smaller numbers of jobs.

Table 1 shows that increases in the number of jobs result in a significant improvement in performance. Also, Table 2 shows that increases in r/P above a value of 0.20 result in a significant improvement in performance. The reason for this is that the partial schedules are identical in Algorithms GAA and XOA, up to the time when the job that is split in XOA starts processing. Therefore, the larger the value of r, the more similar are the costs of the two algorithms for the $\sum w_j C_j$ objective.

5. Approximation Schemes

5.1. Total Weighted Completion Time

This section describes a fully polynomial time approximation scheme (FPTAS) for problem $1 \mid r_R, \Delta_{\max} \leqslant k \mid \sum w_j C_j$. An FPTAS is a family of algorithms $\{A_\epsilon\}$ such that for any $\epsilon > 0$, A_ϵ delivers a schedule that is within a factor $1 + \epsilon$ of optimality and has a running time that is polynomial in n and $1/\epsilon$. An entertaining introduction to FPTAS design is provided by Schuurman and Woeginger (2009).

We start by describing an Algorithm TWCAS$_\epsilon$ that defines our approximation scheme. It is based on an approximate dynamic program. The enumeration within the dynamic program appends a job to the earlier schedule and to the later schedule. The state space is partitioned into boxes, and we approximate the solution by retaining only one state within any box. We assume that the preprocessing step described in §3.2 is applied.

Algorithm TWCAS$_\epsilon$

Initialization. Set $j = h$.

State Variables. (j, t_1, t_2, w, v) corresponds to a partial schedule containing jobs $1, \ldots, j$, where jobs of the earlier schedule occupy the interval $[0, t_1]$, jobs of the later schedule are currently scheduled in the interval $[0, t_2]$ but will subsequently be appended to the earlier schedule, w is the total weight of jobs in the later schedule, and v is the total weighted completion time of all scheduled jobs.

Initial State.

$$\left(h, \sum_{i=1}^{h-1} p_i, p_h, w_h, v_0\right), \quad \text{where } v_0 = \sum_{j=1}^{h-1} w_j \sum_{i=1}^{j} p_i + w_h p_h.$$

Trial State Generation. For each state (j, t_1, t_2, w, v), generate two trial states $(j+1, t_1', t_2', w', v') = (j+1, \max\{t_1, \bar{r}_{j+1}\} + p_{j+1}, t_2, w, v')$ and $(j+1, t_1'', t_2'', w'', v'') = (j+1, t_1, t_2 + p_{j+1}, w + w_{j+1}, v'')$, where $v' = v + w_{j+1}(\max\{t_1, \bar{r}_{j+1}\} + p_{j+1})$ and $v'' = v + w_{j+1}(t_2 + p_{j+1})$. The first of these trial states is only generated when $j+1 \in J\backslash R$, $t_1 < r$, and $\max\{t_1, \bar{r}_{j+1}\} + p_{j+1} \leqslant \bar{d}_h - p_h$.

Trial State Labeling. For each trial state $(j+1, t_1', t_2', w', v')$ and $(j+1, t_1'', t_2'', w'', v'')$, attach the label $(j+1, \Delta(t_2'), \Delta(w'), \Delta(v'))$, and $(j+1, \Delta(t_2''), \Delta(w''), \Delta(v''))$, respectively, where the function Δ is defined by $\Delta(x) = \delta^i$ when a value of x satisfies $\delta^i \leqslant x < \delta^{i+1}$ and $\delta = 1 + \epsilon/(2n)$.

Trial State Elimination. For each pair of trial states with identical labels, eliminate the one with the larger value of its t_1 variable, choosing arbitrarily if the two t_1 values are identical.

Termination Test. If $j+1 < n$, then set $j = j+1$ and return to the trial state generation step. Otherwise, select a state $(n, \hat{t}_1, \hat{t}_2, \hat{w}, \hat{v})$, where $\hat{w}\max\{\hat{t}_1, r\} + \hat{v}$ is smallest, and backtrack to find the corresponding schedule $\hat{\sigma}_\epsilon$.

The state variables in Algorithm TWCAS$_\epsilon$ are defined as (j, t_1, t_2, w, v). The trial state generation step appends job $j+1$ to the earlier and to the later schedule to create two trial states (j, t_1', t_2', w', v') and $(j+1, t_1'', t_2'', w'', v'')$. The trial state labeling step attaches labels $(j+1, \Delta(t_2'), \Delta(w'), \Delta(v'))$ and $(j+1, \Delta(t_2''), \Delta(w''), \Delta(v''))$, where $\Delta(t_2')$, $\Delta(w')$, and $\Delta(v')$ are close to the values of the corresponding state variables t_2', w', and v' (and similarly for $\Delta(t_2'')$, $\Delta(w'')$, and $\Delta(v'')$). On this basis, when two states have the same label, one of them is eliminated in the trial state elimination step. The final part of the termination test step appends the later schedule to the earlier schedule, starting at time $\max\{\hat{t}_1, r\}$, and then selects a state for which the corresponding total weighted completion time is smallest.

Algorithm TWCAS$_\epsilon$ bases its elimination on boxes whose coordinates increase geometrically. The coordinates of the boxes are $(\delta^{m_1}, \delta^{m_2}, \delta^{m_3})$, $(\delta^{m_1+1}, \delta^{m_2}, \delta^{m_3})$, $(\delta^{m_1}, \delta^{m_2+1}, \delta^{m_3})$, $(\delta^{m_1}, \delta^{m_2}, \delta^{m_3+1})$, $(\delta^{m_1+1}, \delta^{m_2+1}, \delta^{m_3})$, $(\delta^{m_1+1}, \delta^{m_2}, \delta^{m_3+1})$, $(\delta^{m_1}, \delta^{m_2+1}, \delta^{m_3+1})$, and $(\delta^{m_1+1}, \delta^{m_2+1}, \delta^{m_3+1})$, for positive integers m_1, m_2, and m_3. Multiplicative errors arise due to mapping each label to the smallest coordinates in its box for state elimination rather than using the actual value of the label. However, as we show below, this produces an overall error of less than δ^n, which is sufficient for our purposes.

Theorem 5. *The family of Algorithms* $\{TWCAS_\epsilon\}$, *for* $\epsilon > 0$, *is an FPTAS for problem* $1 \mid r_R, \Delta_{\max} \leqslant k \mid \sum w_j C_j$, *with* $O((n^4/\epsilon^3)\log P \log W \log(PW))$ *running time.*

Proof. We start by analyzing the cost of the solution delivered by Algorithm TWCAS$_\epsilon$. First, we use induction to establish that the states that remain after applying the trial state elimination step provide a good approximation to those that would be obtained under an exact dynamic program in which the definition of Δ in the trial state labeling step is changed to the identity function defined by $\Delta(x) = x$. Specifically, our induction hypothesis states that given any state (j, t_1, t_2, w, v) that is obtained under the exact dynamic program, a state $(j, \tilde{t}_1, \tilde{t}_2, \tilde{w}, \tilde{v})$ is generated by Algorithm TWCAS$_\epsilon$ with (i) $\tilde{t}_1 \leqslant t_1$, (ii) $\tilde{t}_2 \leqslant \delta^{j-h} t_2$, (iii) $\tilde{w} \leqslant \delta^{j-h} w$, and (iv) $\tilde{v} \leqslant \delta^{j-h} v$. Our proof uses induction on $j-h$. The hypothesis is easily seen to hold for $j-h=1$.

Suppose that the hypothesis holds for $j-h = 1, \ldots, l-1$. Let $(h+l-1, t_1, t_2, w, v)$ define the values of the state variables corresponding to the first $h+l-1$ jobs for some state in the exact dynamic program. First, we assume that job $h+l$ appears in the earlier schedule obtained under the exact dynamic program. In the exact dynamic program, this yields a state $(h+l, t_1', t_2', w', v')$, where $t_1' = \max\{t_1, \bar{r}_{h+l}\} + p_{h+l}$, $t_2' = t_2$, $w' = w$, and $v' = v + w_{h+l}(\max\{t_1, \bar{r}_{h+l}\} + p_{h+l})$. By the induction hypothesis, we have a state with values $(h+l-1, \tilde{t}_1, \tilde{t}_2, \tilde{w}, \tilde{v})$ generated by Algorithm TWCAS$_\epsilon$ for which $\tilde{t}_1 \leqslant t_1$, $\tilde{t}_2 \leqslant \delta^{l-1} t_2$, $\tilde{w} \leqslant \delta^{l-1} w$, and $\tilde{v} \leqslant \delta^{l-1} v$. The trial state resulting from scheduling job $h+l$ in the earlier schedule in Algorithm TWCAS$_\epsilon$ is $(h+l, \tilde{t}_1', \tilde{t}_2', \tilde{w}', \tilde{v}')$, where $\tilde{t}_1' = \max\{\tilde{t}_1, \bar{r}_{h+l}\} + p_{h+l}$, $\tilde{t}_2' = \tilde{t}_2$, $\tilde{w}' = \tilde{w}$, and $\tilde{v}' = \tilde{v} + w_{h+l}(\max\{\tilde{t}_1, \bar{r}_{h+l}\} + p_{h+l})$. However, this trial state may be eliminated by another trial state $(h+l, \tau_1, \tau_2, \omega, \nu)$, where $\tau_1 \leqslant \tilde{t}_1'$, $\tau_2 \leqslant \delta\tilde{t}_2'$, $\omega \leqslant \delta\tilde{w}'$, and $\nu \leqslant \delta\tilde{v}'$, because both of these trial states may have the same label. For the trial state $(h+l, \tau_1, \tau_2, \omega, \nu)$, we have

(i) $\tau_1 \leqslant \tilde{t}_1' = \max\{\tilde{t}_1, \bar{r}_{h+l}\} + p_{h+l} \leqslant \max\{t_1, \bar{r}_{h+l}\} + p_{h+l} = t_1'$;

(ii) $\tau_2 \leqslant \delta\tilde{t}_2' = \delta\tilde{t}_2 \leqslant \delta^l t_2 = \delta^l t_2'$;

(iii) $\omega \leqslant \delta\tilde{w}' = \delta\tilde{w} \leqslant \delta^l w = \delta^l w'$;

(iv) $\nu \leqslant \delta\tilde{v}' = \delta\tilde{v} + \delta w_{h+l}(\max\{\tilde{t}_1, \bar{r}_{h+l}\} + p_{h+l}) < \delta^l v + \delta^l w_{h+l}(\max\{t_1, \bar{r}_{h+l}\} + p_{h+l}) = \delta^l v'$.

This establishes that the induction hypothesis holds for $j = h+l$ when job $h+l$ appears in the earlier schedule obtained under the exact dynamic program.

We now consider the alternative case where job $h+l$ appears in the later schedule obtained under the exact dynamic program. In the exact dynamic program, this yields a state $(h+l, t_1'', t_2'', w'', v'')$, where $t_1'' = t_1$, $t_2'' = t_2 + p_{h+l}$, $w'' = w + w_{h+l}$, and $v'' = v + w_{h+l}(t_2 + p_{h+l})$. By the induction hypothesis, we have a state $(h+l-1, \tilde{t}_1, \tilde{t}_2, \tilde{w}, \tilde{v})$ in the approximate dynamic program for which $\tilde{t}_1 \leqslant t_1$, $\tilde{t}_2 \leqslant \delta^{l-1} t_2$, $\tilde{w} \leqslant \delta^{l-1} w$, and $\tilde{v} \leqslant \delta^{l-1} v$. The trial state resulting from scheduling job $h+l$ in the later schedule in Algorithm TWCAS$_\epsilon$ is $(h+l, \tilde{t}_1'', \tilde{t}_2'', \tilde{w}'', \tilde{v}'')$, where $\tilde{t}_1'' = \tilde{t}_1$, $\tilde{t}_2'' = \tilde{t}_2 + p_{h+l}$, $\tilde{w}'' = \tilde{w} + w_{h+l}$, and $\tilde{v}'' = \tilde{v} + w_{h+l}(\tilde{t}_2 + p_{h+l})$. However, this trial state may be eliminated by another trial state $(h+l, \tau_1, \tau_2, \omega, \nu)$, where $\tau_1 \leqslant \tilde{t}_1''$, $\tau_2 \leqslant \delta \tilde{t}_2''$, $\omega \leqslant \delta \tilde{w}''$, and $\nu \leqslant \delta \tilde{v}''$, because both of these trial states may have the same label. For the trial state $(h+l, \tau_1, \tau_2, \omega, \nu)$, we have:

(i) $\tau_1 \leqslant \tilde{t}_1'' = \tilde{t}_1 \leqslant t_1 = t_1''$;

(ii) $\tau_2 \leqslant \delta \tilde{t}_2'' = \delta \tilde{t}_2 + \delta p_{h+l} < \delta^l t_2 + \delta^l p_{h+l} = \delta^l t_2''$;

(iii) $\omega \leqslant \delta \tilde{w}'' = \delta \tilde{w} + \delta w_{h+l} < \delta^l w + \delta^l w_{h+l} = \delta^l w''$;

(iv) $\nu \leqslant \delta \tilde{v}'' = \delta \tilde{v} + \delta w_{h+l}(\tilde{t}_2 + p_{h+l}) < \delta^l v + \delta^l w_{h+l}(t_2 + p_{h+l}) = \delta^l v''$.

This establishes that the induction hypothesis holds for $j = h+l$ when job $h+l$ appears in the later schedule obtained under the exact dynamic program. Thus, the induction hypothesis holds in each case.

We now use the induction hypothesis to show that Algorithm TWCAS$_\epsilon$ delivers a solution with the desired worst-case bound. Let the state (n, t_1, t_2, w, v) correspond to an optimal solution in the exact dynamic program, which yields a solution value $\sum w_j C_j(\sigma^*) = w \max\{t_1, r\} + v$. Then the above induction argument shows that Algorithm TWCAS$_\epsilon$ generates a state $(n, \tilde{t}_1, \tilde{t}_2, \tilde{w}, \tilde{v})$ with a solution value $\tilde{w} \max\{\tilde{t}_1, r\} + \tilde{v}$, where $\tilde{t}_1 \leqslant t_1$, $\tilde{w} \leqslant \delta^{n-h} w$, and $\tilde{v} \leqslant \delta^{n-h} v$. Using these inequalities, we obtain $\tilde{w} \max\{\tilde{t}_1, r\} + \tilde{v} \leqslant \delta^{n-h} w \max\{t_1, r\} + \delta^{n-h} v < \delta^n (w \max\{t_1, r\} + v)$. We now show that $\delta^n \leqslant 1 + \epsilon$, for $0 \leqslant \epsilon \leqslant 2$. This inequality follows because $\delta^n = (1 + \epsilon/(2n))^n$ is a convex function of ϵ, whereas $1+\epsilon$ is linear, and the inequality holds at the ends of the range when $\epsilon = 0$ and $\epsilon = 2$. Thus, we have established the desired inequality $\sum w_j C_j(\hat{\sigma}_\epsilon) \leqslant (1+\epsilon) \sum w_j C_j(\sigma^*)$.

It remains to analyze the time complexity of Algorithm TWCAS$_\epsilon$. Because at most one state is retained for each possible label after the trial state elimination step, and each state generates at most two trial states, the maximum number of retained states is twice the number of labels. The labels are of the form $(j+1, \Delta(t_2), \Delta(w), \Delta(v))$, where $j \leqslant n$, $t_2 \leqslant P$, $w \leqslant W$, and $v \leqslant 2PW$ because $2PW$ is an upper bound on the optimal solution value. The number of possible values of $\Delta(t_2)$ is given by

$$L = \lceil \log_\delta P \rceil = \lceil \ln P / \ln \delta \rceil \leqslant \lceil (1 + 2n/\epsilon) \ln P \rceil,$$

where the last inequality is obtained from $\delta = 1 + \epsilon/(2n)$ and the well-known inequality $\ln x \geqslant (x-1)/x$ for all $x \geqslant 1$. Similarly, the number of values of $\Delta(w)$ is bounded above by $(1 + 2n/\epsilon) \ln W$, and the number of values of $\Delta(v)$ is bounded above by $(1 + 2n/\epsilon) \ln(2PW)$. Thus, the overall time complexity of Algorithm TWCAS$_\epsilon$ is $O((n^4/\epsilon^3) \log P \log W \log(PW))$. $\square$

5.2. Other Objectives

The online companion contains a detailed description of fully polynomial time approximation schemes for the other three scheduling objectives. We now provide an overview of that analysis.

For the maximum lateness problem, we again adopt the assumption that all due dates are nonpositive. The state variables are (j, t_1, t_2, v), where v denotes the maximum lateness of the jobs in the later schedule. In the trial state generation step, the values of v' and v'' are given by $v' = v$ and $v'' = \max\{v, t_2 + p_{j+1} - d_{j+1}\}$. In the trial state labeling step, the label associated with a state $(j+1, t_1', t_2', v')$ is $(j+1, \lfloor t_2'/\delta_1 \rfloor \delta_1, \lfloor v/\delta_2 \rfloor \delta_2)$, where $\delta_1 = \epsilon P/2n$, and $\delta_2 = \epsilon L_{\max}(\pi^*)/2$. For a given value of j, Algorithm LAS$_\epsilon$ assigns the trial states to equal-sized boxes according to the values of (t_2', v') and (t_2'', v''), where each box has coordinates $(m_1\delta_1, m_2\delta_2)$, $((m_1+1)\delta_1, m_2\delta_2)$, $(m_1\delta_1, (m_2+1)\delta_2)$, and $((m_1+1)\delta_1, (m_2+1)\delta_2)$, for nonnegative integers m_1 and m_2. For any two states with the same value of j and where the last two components of their labels assign them to the same box, one of these states is eliminated. However, additive approximation errors arise through this approach of mapping each label to the smallest coordinates in its box for state elimination rather than using the actual value of the label. For t_2', there is an error of δ_1, but this error does not accumulate across iterations. For t_2'', an error of at most δ_1 arises at each iteration, giving an overall error of at most $n\delta_1$. Similarly, for v', there is an error of δ_2 that does not accumulate, whereas for v'', an error of at most $n\delta_1$ occurs when v'' is determined by $t_2 + p_{j+1} - d_{j+1}$ together with an additional error of at most δ_2. Thus, the overall error for v' or v'' is at most $n\delta_1 + \delta_2 = \epsilon(P + L_{\max}(\pi^*))/2 \leqslant \epsilon L_{\max}(\sigma^*)$, and the termination test step shows that this is the overall approximation error.

For the makespan problem, we use the same approximation scheme as for the maximum lateness problem. However, the variables t_2 and v become identical, so v is discarded. The analysis of the algorithm is similar to that for the maximum lateness problem.

Finally, we consider the total completion time problem. We can use Algorithm TWCAS$_\epsilon$ as a basis for an approximation scheme. However, an alternative is to use a similar rounding approach for computing labels to the one outlined above for the maximum lateness problem, which produces equal-sized boxes for trial state elimination. The state variables are (j, t_1, t_2, w, v), where w denotes the number of jobs in the later schedule. In the trial state generation step, the values of w', v', w'', and v'' are given by $w' = w$, $v' = v + \max\{t_1, \bar{r}_{j+1}\} + p_{j+1}$, $w'' = w + 1$, and $v'' = v + t_2 + p_{j+1}$. In the trial state labeling step, the label associated with a state $(j+1, t_1', t_2', w', v')$ is $(j+1, \lfloor t_2'/\delta_1 \rfloor \delta_1, w', \lfloor v/\delta_2 \rfloor \delta_2)$, where $\delta_1 = \epsilon P/2n^2$ and $\delta_2 = \epsilon P/2n$. The analysis of the

Table 3. Results for maximum time deviation rescheduling problems.

Objective	Optimal algorithm	NP-completeness	Worst-case ratio bound	Approximation scheme
$\sum w_j C_j$	$O(nPTW)$	*BNPC*	2	$O((n^4/\epsilon^3)\log P \cdot \log W \log(PW))$
$C_{\max}$	$O(nT)$	*BNPC*	2	$O(n^2/\epsilon)$
$L_{\max}$	$O(nPT)$	*BNPC*	2	$O(n^2/\epsilon^2)$
$\sum C_j$	$O(n^2PT)$	*BNPC*	5/3	$O(n^6/\epsilon^2)$

Note. $T \leqslant P$.

algorithm is also similar to that for the maximum lateness problem.

6. Concluding Remarks

This paper considers the issue of rescheduling to allow for unexpected delays in the availability of new jobs, taking into account the effect of this disruption on a previously planned optimal schedule. The effect of disruption is measured by the change in completion times for the jobs, relative to the planned schedule. We consider the classical objective of minimizing the total weighted completion time for the scheduling cost, and model the amount of schedule change as a constraint. We provide an optimal solution algorithm that runs in reasonable computing time, a proof of binary NP-completeness of the recognition version of the problem, a linear time approximation algorithm with an associated worst-case ratio bound that is fast enough for practical implementation, and an approximation scheme. Three other classical scheduling objectives are also similarly considered. Our results are summarized in Table 3, where *BNPC* indicates that the recognition version of the problem is binary NP-complete.

Several important issues remain open for future research. First, many of our results may be extendable to a more general problem in which the initially unavailable jobs have various release dates. Second, it would be valuable to extend our work to consider other classical definitions of scheduling cost. Third, there are other relevant measures of schedule change that also require examination. For example, it is shown in the online companion that deciding whether problem $1 \mid r_R, \sum \Delta_j \leqslant k \mid \cdot$, i.e., with no objective function specified, has a feasible schedule is unary NP-complete. Fourth, our work needs to be extended to include other manufacturing environments such as flow shops and job shops. Fifth, various disruptions besides those considered here and in Hall and Potts (2004) occur frequently in practice and need to be studied. We hope that our work will encourage further research that will mitigate the effects of the disruptions that occur frequently in manufacturing practice.

7. Electronic Companion

An electronic companion to this paper is available as part of the online version that can be found at http://or.journal.informs.org/.

Acknowledgments

This work is supported in part by National Science Foundation grants DMI-9821033 and DMI-0421823, by the Summer Fellowship Program of the Fisher College of Business (The Ohio State University), by the CORMSIS Visitor Programme (University of Southampton, United Kingdom), and by EPSRC grants GR/R97696/01 and EP/D060931/1. Expert programming assistance was provided by Zhixin Liu (The Ohio State University).

References

Aytug, H., M. A. Lawley, K. N. McKay, S. Mohan, R. Uzsoy. 2005. Executing production schedules in the face of uncertainties: A review and some future directions. *Eur. J. Oper. Res.* **161** 86–110.

Bean, J. C., J. R. Birge, J. Mittenthal, C. E. Noon. 1991. Matchup scheduling with multiple resources, release dates and disruptions. *Oper. Res.* **39** 470–483.

Belouadah, H., M. E. Posner, C. N. Potts. 1992. Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete Appl. Math.* **36** 213–231.

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

Davenport, A. J., J. C. Beck. 2000. A survey of techniques for scheduling under uncertainty. Working paper, IBM T. J. Watson Research Center, Yorktown Heights, NY.

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

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

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

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

Herroelen, W., R. Leus. 2005. Project scheduling under uncertainty: Survey and research potentials. *Eur. J. Oper. Res.* **165** 289–306.

Labetoulle, J., E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan. 1984. Preemptive scheduling of uniform machines subject to release dates. W. R. Pulleyblank, ed. *Progress in Combinatorial Optimization*. Academic Press, New York, 245–261.

Rinnooy Kan, A. H. G. 1976. *Machine Scheduling Problems: Classification, Complexity and Computations*. Nijhoff, The Hague, The Netherlands.

Schuurman, P., G. Woeginger. 2009. Approximation schemes—A tutorial. R. H. Möhring, C. N. Potts, A. S. Schulz, G. J. Woeginger, L. A. Wolsey, eds. *Lectures on Scheduling*. Forthcoming.

Smith, W. E. 1956. Various optimizers for single-stage production. *Naval Res. Logist. Quart.* **3** 59–66.

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

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

Wu, S. D., R. H. Storer, P.-C. Chang. 1992. A rescheduling procedure for manufacturing systems under random disruptions. G. Fandel, T. Gulledge, A. Jones, eds. *New Directions for Operations Research in Manufacturing*. Springer, Berlin, 292–308.

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.

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