INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Rescheduling for Multiple New Orders

Nicholas G. Hall, Zhixin Liu, Chris N. Potts,

To cite this article:
Nicholas G. Hall, Zhixin Liu, Chris N. Potts, (2007) Rescheduling for Multiple New Orders. INFORMS Journal on Computing 19(4):633-645. https://doi.org/10.1287/ijoc.1060.0209

INFORMS Journal on Computing
Vol. 19, No. 4, Fall 2007, pp. 633–645
ISSN 1091-9856 | EISSN 1526-5528 | 07 | 1904 | 0633

DOI 10.1287/ijoc.1060.0209

Rescheduling for Multiple New Orders

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

Zhixin Liu
Department of Management Studies, School of Management, University of Michigan–Dearborn, Dearborn, Michigan 48126, zhixin@umd.umich.edu

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

A set of original jobs has been scheduled on a single machine, but not processed, when a set of new jobs arrives. The decision maker needs to insert the new jobs into the existing schedule without excessively changing it. The objective is minimization of the maximum lateness of the jobs, subject to a customer service requirement modeled by a limit on the maximum time change of the original jobs. Because the schedule of the original jobs can be arbitrary, this problem models multiple disruptions from repeated new job arrivals. We show that this scheduling problem is intractable, even if no new jobs arrive. We describe several approximation algorithms and analyze their worst-case performance. Next, we develop a branch and bound algorithm that uses a variable neighborhood descent algorithm to obtain an initial upper bound, several dominance properties that we establish, and a lower bounding scheme based on a preemptive relaxation of the problem. The branch and bound algorithm solves 99.9% of randomly generated instances with up to 1,000 jobs within 60 CPU seconds. Our work demonstrates for the first time that optimization of large scale, intractable rescheduling problems is possible. More generally, it refocuses the literature on scheduling problems towards rescheduling issues.

Key words: deterministic scheduling; rescheduling for new job disruptions; heuristic worst-case analysis; branch and bound algorithm

History: Accepted by William J. Cook, Area Editor for Design and Analysis of Algorithms; received March 2005; revised August 2005, April 2006, August 2006; accepted August 2006. Published online in *Articles in Advance* September 28, 2007.

1. Introduction

In recent years, *rescheduling* has attracted considerable attention, 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. In dynamic environments, managers and production planners must not only generate high quality schedules, but also react quickly to unexpected events and revise their schedules in a cost-effective manner (Vieira et al. 2003).

In general, there are many possible *disruptions* that can make rescheduling necessary, including the arrival of new orders, machine breakdowns, shortage of materials or resources, larger or smaller than expected processing times, cancellation of orders, changes in order priority, and due date changes.

There are several well-documented real-world applications of rescheduling. Bean et al. (1991) consider an automobile industry application, and propose a matchup scheduling approach that compensates for a disruption. Zweben et al. (1993) describe the GERRY scheduling and rescheduling system that supports the space shuttle using iterative repair heuristics. Clausen et al. (2001) describe a shipyard application, where the goal of rescheduling is to store large steel plates for efficient access. A similar problem occurs in the assignment of stacker cranes to berths at container ports. Yu et al. (2003), in their work that received the 2002 Edelman Award of INFORMS (http://www.informs.org/Prizes/EdelmanPrizeDetails.html), discuss a short-range airline planning problem. Their optimization-based approach for rescheduling to compensate for air traffic, weather, crew unavailability, and other disruptions, helped Continental Airlines recover after the terrorist attacks of September 11, 2001.

Because of the practical importance of rescheduling, a number of researchers propose rescheduling approaches for a variety of scheduling environments. Szelke and Kerr (1994), Davenport and Beck (2000), Herroelen and Leus (2005), Vieira et al. (2003), and Aytug et al. (2004) provide extensive reviews of the rescheduling literature, including taxonomies, strategies and algorithms, in both deterministic and stochastic environments.

Wu et al. (1993) suggest a composite objective rescheduling approach for a single machine problem, and design heuristic procedures for solving it. Their criteria include minimizing the makespan and the impact of the schedule change. Unal et al. (1997) consider a single machine problem with newly arrived jobs with setup times that depend on part types. They consider inserting new jobs into the original schedule to minimize the total weighted completion time, or makespan, of the new jobs, without incurring additional setups or causing schedule delays. However, these constraints may be too restrictive in practice. Hall and Potts (2007) consider problems where a set of jobs becomes available later than expected, after a schedule has been determined to minimize a classical cost objective. In the new schedule, the limit on allowable disruption is measured by the maximum time disruption to any of the jobs between the original and adjusted schedules.

In this paper we discuss the following rescheduling problem. A set of *original* jobs has been scheduled to minimize a given cost objective, and then several sets of *new* jobs arrive unexpectedly. The objective in the original schedule still needs to be minimized over all of the jobs. However, this will change the original schedule, reducing customer satisfaction and creating havoc with the original resource allocations. Thus, the tradeoff between the scheduling cost and the disruption cost must be considered in detail. The problem of job arrivals that can be modeled as a single new order is studied by Hall and Potts (2004) who measure the disruption either by the amount of resequencing or by the amount of time disruption. In either case this disruption is treated both as a constraint and as a cost.

In many practical problems however, consideration of a single new order, as in Hall and Potts (2004), is not general enough. Complex and dynamic scheduling environments may generate disruptions that occur at different times and cannot be viewed as a single new order. Therefore, we consider multiple arriving new orders. Here, as a result of previous disruptions, it is not necessarily the case that the original schedule optimizes the given scheduling measure. Therefore, important properties of the original schedule are lost, and the problem becomes significantly more difficult to solve.

In Section 2, we introduce our notation, formally define the rescheduling problem, and present a computational complexity result. Section 3 analyzes the worst-case performance of several specific classes of schedules and of an approximation algorithm. In Section 4, we propose lower and upper bounding schemes, and establish dominance properties that are then integrated into a branch and bound algorithm. In Section 5, we provide an extensive computational study of this algorithm. Finally, Section 6 contains a summary of the paper, along with suggestions for future research.

2. Problem Definition and Computational Complexity

In this section, we give a formal definition of the rescheduling problem and then discuss its computational complexity.

Let $J_O = \{1, \ldots, n_O\}$ denote a set of original jobs to be processed nonpreemptively on a single machine. Let π^* denote a schedule that minimizes the lateness of the latest job in J_O. From Jackson (1955), a possible construction for π^* is to sequence the jobs in nondecreasing order of their due dates (EDD order) and schedule them without idle time between the jobs. Let π denote an arbitrary schedule for J_O, possibly including some idle time between the jobs. Also, let J_N denote a set of new jobs, where $n_N = |J_N|$. We assume that all the new jobs arrive at time zero, after a schedule for the jobs of J_O has been determined, but before processing begins. There is no loss of generality in this assumption: if the jobs arrive after time zero, then the processed jobs of J_O are removed from the problem, any partly processed jobs are processed to completion, and J_O and n_O are updated accordingly.

Let $J = J_O \cup J_N$, and $n = n_O + n_N$. Also, let p_j denote the positive processing time of job j, and let d_j denote the due date of job j (which can be negative), for $j \in J$.

Let $\nu \in \{\pi^*, \pi\}$. For any schedule σ of the jobs of J, we define the following variables:

$C_j(\sigma) =$ the *completion time* of job j, for $j \in J$;
$L_j(\sigma) = C_j(\sigma) - d_j$, the *lateness* of job j, for $j \in J$;
$\Delta_j(\nu, \sigma) = |C_j(\sigma) - C_j(\nu)|$, the *time disruption* of job j, for $j \in J_O$,

where the time disruption of job j in schedule σ is the absolute value of the difference between the completion time of that job in σ and ν. When there is no ambiguity, we simplify $C_j(\sigma)$, $L_j(\sigma)$, and $\Delta_j(\nu, \sigma)$ to C_j, L_j, and $\Delta_j(\nu)$, respectively. Let $C_{\max} = \max_{j \in J}\{C_j\}$, $L_{\max} = \max_{j \in J}\{L_j\}$, and $\Delta_{\max}(\nu) = \max_{j \in J_O}\{\Delta_j(\nu)\}$. The time disruption measure models penalties associated with the change of delivery times of jobs to customers, and the cost of rescheduling the resources so that they are available at the new times when they are required.

Let k denote the given upper bound on the allowed time disruption of any original job. Following the standard $\alpha|\beta|\gamma$ classification scheme for scheduling problems (Graham et al. 1979), α indicates the scheduling environment, β describes the job characteristics or restrictive requirements, and γ specifies the objective function to be minimized. We consider only single machine problems, thus implying that $\alpha = 1$. Under β, we use $\Delta_{\max}(\pi^*) \le k$ or $\Delta_{\max}(\pi) \le k$ to describe a constraint on the maximum time disruption for any job of J_O. We also use *pmtn* to

denote a schedule in which preemption of jobs is allowed. Finally, the objective is to minimize the maximum lateness, $L_{\max}$. Thus, our problems are denoted by $1|\Delta_{\max}(\pi^*) \le k|L_{\max}$ and $1|\Delta_{\max}(\pi) \le k|L_{\max}$. Problem $1|\Delta_{\max}(\pi) \le k|L_{\max}$ is more general than problem $1|\Delta_{\max}(\pi^*) \le k|L_{\max}$, and as we discuss below much less tractable. For problem $1|\Delta_{\max}(\pi) \le k|L_{\max}$, which is the main focus of this paper, the constraint $\Delta_{\max}(\pi) \le k$ implies that $C_j \ge C_j(\pi) - k$ and $C_j \le C_j(\pi) + k$. Thus, each job j, for $j \in J_O$, has an *implied release date* $\bar{r}_j = C_j(\pi) - p_j - k$ specifying the earliest time at which it can start processing, and an *implied deadline* $\bar{d}_j = C_j(\pi) + k$ specifying the time by which it must be completed; $\bar{r}_j = 0$ and $\bar{d}_j = \infty$ for $j \in J_N$.

It is important to note that, as a result of previous disruptions, the original schedule, denoted by π, may no longer be optimal with respect to the scheduling cost. Consequently, we allow the original schedule to be arbitrary, possibly containing machine idle time. Thus, $1|\Delta_{\max}(\pi) \le k|L_{\max}$ corresponds to the multiple-disruption rescheduling problem.

Hall and Potts (2004) present an $O(n + n_N \log n_N)$ time algorithm to solve problem $1|\Delta_{\max}(\pi^*) \le k|L_{\max}$. They also prove that the recognition version of problem $1|\Delta_{\max}(\pi) \le k|L_{\max}$ for arbitrary π is unary NP-complete, which refers to problems for which no polynomial time algorithm can exist, even if the data are unary encoded, unless P = NP (Garey and Johnson 1979). We now provide a new proof of this computational complexity result, which shows that the same result holds even if no new jobs arrive.

THEOREM 1. *The recognition version of problem* $1|\Delta_{\max}(\pi) \le k|L_{\max}$ *with* $J_N = \varnothing$ *is unary* NP*-complete.*

PROOF. By reduction from the following problem, which is known to be unary NP-complete.

3-Partition (Garey and Johnson 1979). Given $3t$ elements with integer sizes $a_1, \ldots, a_{3t}$, where $\sum_{i=1}^{3t} a_i = ty$ and $y/4 < a_i < y/2$ for $i = 1, \ldots, 3t$, does there exist a partition $S_1, \ldots, S_t$ of the index set $\{1, \ldots, 3t\}$ such that $|S_j| = 3$ and $\sum_{i \in S_j} a_i = y$ for $j = 1, \ldots, t$?

Consider the following instance of the recognition version of problem $1|\Delta_{\max}(\pi) \le k|L_{\max}$: $n_O = 4t - 1$; $p_i = a_i$ and $d_i = ty + t - 1$ for $i = 1, \ldots, 3t$; $p_i = 1$ and $d_i = (i - 3t)y + i - 3t$ for $i = 3t + 1, \ldots, 4t - 1$; $k = (t-1)y$; and $C = 0$, where C is a threshold value for $L_{\max}$. In schedule π, jobs $1, \ldots, 3t$ are scheduled within the interval $[0, ty]$ in arbitrary order without idle time, jobs $3t+1, \ldots, 4t-1$ are scheduled within the intervals $[(i-2t-1)y + i - 3t - 1, (i-2t-1)y + i - 3t]$, for $i = 3t+1, \ldots, 4t-1$, and idle time appears between the final $t-1$ intervals.

We prove that there exists a feasible schedule for this instance of $1|\Delta_{\max}(\pi) \le k|L_{\max}$ with $L_{\max} \le C$ if and only if there exists a solution to 3-Partition.

($\Rightarrow$) Consider a no-idle schedule σ in which job $3t+1$ is processed in the interval $[y, y+1]$, job $3t+2$ is processed in the interval $[2y+1, 2y+2], \ldots$, and job $4t-1$ is processed in the interval $[(t-1)y + t - 2, (t-1)y + t - 1]$. Then there are t intervals of length y, including the one starting at time $(t-1)y + t - 1$. We process jobs $1, \ldots, 3t$ in these t intervals. In each interval, there are exactly three jobs with total processing time y, and they are processed in the same order as in π. Also, the jobs in the first position in each interval are processed in the same order as in π. It follows that $L_{\max} \le C$. It is also clear that for the jobs $3t+1, \ldots, 4t-1$, we have $\Delta_{\max} = (t-1)y$. Now we consider the jobs in $\{1, \ldots, 3t\}$. First, if a job moves earlier from π to σ, the move is by no more than from ty in π to y in σ (which could possibly occur for the last job of π); thus, the time disruption is no greater than $(t-1)y$. Second, if a job moves later from π to σ, this move is no more than from $(t-1)(y/4+1)$ in π to $(t-1)y + t - 1$ in σ (which could possibly occur for the job that starts at time $(t-1)y + t - 1$ in σ); therefore, the time disruption in this case is less than $(t-1)y$. Thus, we obtain a feasible schedule with $L_{\max} \le C$.

($\Leftarrow$) A feasible schedule σ with $L_{\max} \le C$ implies that job $3t+1$ completes processing no later than time $y+1$, so since $\Delta_{\max}(\pi) \le (t-1)y$, it must be processed in the interval $[y, y+1]$. Similarly, job $3t+2$ must be processed in the interval $[y+1, y+2], \ldots$, and job $4t-1$ must be processed in the interval $[(t-1)y + t - 2, (t-1)y + t - 1]$. Moreover, each of the jobs $1, \ldots, 3t$ must complete no later than time $ty + t - 1$. Therefore, there is no idle time in the schedule, and three jobs in each remaining interval have total processing time of exactly y. Therefore, there exists a solution to 3-Partition. □

Let σ^* denote an optimal schedule for problem $1|\Delta_{\max}(\pi) \le k|L_{\max}$.

COROLLARY 1. *Problem* $1|\Delta_{\max}(\pi) \le k|L_{\max}$ *does not have a polynomial time approximation algorithm delivering a schedule* σ *with* $L_{\max}(\sigma) \le \rho L_{\max}(\sigma^*)$ *for all instances for any finite* ρ*, unless* P = NP.

PROOF. Since the threshold $L_{\max}$ value used in the proof of Theorem 1 is zero, the result follows immediately. □

Henceforth, we restrict our discussion to problem $1|\Delta_{\max}(\pi) \le k|L_{\max}$.

3. Approximation

This section focuses on the worst-case performance of approximation algorithms. All results are proved under the standard assumption that $d_j \le 0$, for $j \in J$. If σ is a specific class of schedule or the schedule obtained from some approximation algorithm, then we establish an inequality $L_{\max}(\sigma) \le \rho L_{\max}(\sigma^*)$ that holds for all problem instances, where ρ is a constant. We refer to ρ as a *performance bound* for schedules of this class or for the approximation algorithm.

Section 3.1 analyzes the worst-case performance of several specific classes of schedules. In Section 3.2, we design an approximation algorithm, analyze its worst-case performance, and show that the resulting performance bound is best possible.

3.1. Analysis of Specific Classes of Schedules

Recall that an *active schedule* is a schedule in which no job can be scheduled earlier without violating a constraint, while a *semiactive schedule* is a schedule in which no job can be moved earlier without changing the sequence or violating a constraint (Brucker 1998). Note that an active schedule is also semiactive. We first analyze the worst-case performance of an arbitrary active or semiactive schedule.

THEOREM 2. *Let σ^{AS} be an arbitrary active or semiactive schedule. Then $L_{max}(\sigma^{AS}) \le 3L_{max}(\sigma^*)$, and no better performance bound exists for an arbitrary active or semiactive schedule.*

PROOF. Consider job j for which $L_j(\sigma^{AS}) = L_{max}(\sigma^{AS})$, and let P_j denote the total processing of jobs that are completed by time $C_j(\sigma^{AS})$. If there is no idle time before job j in σ^{AS}, then

$$L_{max}(\sigma^{AS}) = P_j - d_j. \tag{1}$$

Alternatively, the last period of idle time before job j occurs immediately before some other job i, where job i starts at time r_i because σ^{AS} is an active or semiactive schedule. Therefore,

$$L_{max}(\sigma^{AS}) \le r_i + P_j - d_j. \tag{2}$$

Using our assumption that $d_j \le 0$ for all jobs j, we obtain $r_i \le C_{max}(\sigma^*) \le L_{max}(\sigma^*)$, $P_j \le C_{max}(\sigma^*) \le L_{max}(\sigma^*)$, and $-d_j \le L_{max}(\sigma^*)$. Substitution in (1) and (2) yields the desired inequality $L_{max}(\sigma^{AS}) \le 3L_{max}(\sigma^*)$.

The following instance shows that no better bound exists.

EXAMPLE 1. $n_O = n-2$, $n_N = 2$, where $n \ge 3$; processing times and due dates are shown in Table 1; $k = (n - 1 + 1/n^2)/2$; and the intervals within which each job is scheduled in π are also given in Table 1.

Schedules σ^{AS} and σ^* are defined in Table 1, with the values of $\Delta_j(\pi, \sigma^{AS})$ and $\Delta_j(\pi, \sigma^*)$ for $j \in J_O$ also given. Precise values for $\Delta_j(\pi, \sigma^*)$ depend on whether n is odd or even. For odd n, $\Delta_j(\pi, \sigma^*)$ is decreasing in j until $j = (n+1)/2$ when $\Delta_{(n+1)/2}(\pi, \sigma^*) = 0 + (n-1)/(2n^2)$, and then is increasing in j starting with $\Delta_{(n+3)/2}(\pi, \sigma^*) = 1 - (n+1)/(2n^2)$. Similarly, for even n, $\Delta_j(\pi, \sigma^*)$ is decreasing in j until $j = n/2$ when $\Delta_{n/2}(\pi, \sigma^*) = 1/2 + (n-2)/(2n^2)$, and then an increasing pattern starts with $\Delta_{n/2+1}(\pi, \sigma^*) = 1/2 - n/(2n^2)$. We compute the solution values $L_{max}(\sigma^{AS}) =$

Table 1 Data for Example 1

Job j	1	2	…	$n-2$	$n-1$	n
p_j	$1/n^2$	$1/n^2$	…	$1/n^2$	$n-1$	1
d_j	0	0	…	0	0	$-n+1$
π	$[(n+1)/2, (n+1)/2 + 1/n^2]$	$[(n+3)/2, (n+3)/2 + 1/n^2]$	…	$[(3n-5)/2, (3n-5)/2 + 1/n^2]$		
σ^{AS}	$[1 - 1/(2n^2), 1 + 1/(2n^2)]$	$[2 - 1/(2n^2), 2 + 1/(2n^2)]$	…	$[n - 2 - 1/(2n^2), n - 2 + 1/(2n^2)]$	$[n - 2 + 1/(2n^2), 2n - 3 + 1/(2n^2)]$	$[2n - 3 + 1/(2n^2), 2n - 2 + 1/(2n^2)]$
$\Delta_j(\pi, \sigma^{AS})$	$(n - 1 + 1/n^2)/2$	$(n - 1 + 1/n^2)/2$	…	$(n - 1 + 1/n^2)/2$		
σ^*	$[n, n + 1/n^2]$	$[n + 1/n^2, n + 2/n^2]$	…	$[n + (n-3)/n^2, n + (n-2)/n^2]$	$[1, n]$	$[0, 1]$
$\Delta_j(\pi, \sigma^*)$	$(n-1)/2$	$(n-3)/2 + 1/n^2$	…	$(n-5)/2 - (n-3)/n^2$		

Table 2 Data for Example 2

Job j	1	2	3	4	5
p_j	v	1	v	1	1
d_j	0	$-2v-2$	0	0	0
π	$[v+1, 2v+1]$	$[2v+1, 2v+2]$	$[2v+2, 3v+2]$	$[4v+1, 4v+2]$	$[4v+2, 4v+3]$
σ^{LO}	$[3v+2, 4v+2]$	$[4v+2, 4v+3]$	$[2v+2, 3v+2]$	$[2v, 2v+1]$	$[2v+1, 2v+2]$
$\Delta_j(\pi, \sigma^{\mathrm{LO}})$	$2v+1$	$2v+1$	0	$2v+1$	$2v+1$
σ^*	$[1, v+1]$	$[0, 1]$	$[v+1, 2v+1]$	$[2v+1, 2v+2]$	$[2v+2, 2v+3]$
$\Delta_j(\pi, \sigma^*)$	v	$2v+1$	$v+1$	$2v$	$2v$

$3n-3+1/(2n^2)$ and $L_{\max}(\sigma^*) = n + (n-2)/n^2$. Therefore, $\lim_{n\to\infty}(L_{\max}(\sigma^{\mathrm{AS}})/L_{\max}(\sigma^*)) = \lim_{n\to\infty}((3n-3+1/(2n^2))/(n+(n-2)/n^2)) = 3$. □

A *locally optimal schedule* is a semiactive schedule in which exchanging the order of any two adjacent jobs (a) will not decrease the maximum lateness of all the jobs, and (b) will not decrease the maximum lateness of these two jobs under the condition that it does not increase the maximum lateness of all the jobs. The following result considers the worst-case performance of locally optimal schedules.

Theorem 3. *Let σ^{LO} denote a locally optimal schedule. Then $L_{\max}(\sigma^{\mathrm{LO}}) \le 3L_{\max}(\sigma^*)$, and no better performance bound exists for an arbitrary locally optimal schedule.*

Proof. Since a locally optimal schedule is by definition semiactive, the worst-case performance bound follows from Theorem 2.

The following instance shows that no better bound exists.

Example 2. $n = n_O = 5$; processing times and due dates are shown in Table 2, where $v \ge 1$; $k = 2v+1$; and the intervals within which each job is scheduled in π are also given in Table 2.

Schedules σ^{LO} and σ^* are defined in Table 2, with the values of $\Delta_j(\pi, \sigma^{\mathrm{LO}})$ and $\Delta_j(\pi, \sigma^*)$ for $j \in J_O$ also given. We therefore obtain $L_{\max}(\sigma^{\mathrm{LO}}) = 6v+5$ and $L_{\max}(\sigma^*) = 2v+3$. Thus, $\lim_{v\to\infty}(L_{\max}(\sigma^{\mathrm{LO}})/L_{\max}(\sigma^*)) = \lim_{v\to\infty}((6v+5)/(2v+3)) = 3$. □

Finally, we analyze the worst-case performance of an arbitrary *no idle time schedule*, which is a schedule with processing starting at time zero and with no idle time between jobs.

Theorem 4. *Let σ^{NI} denote an arbitrary no idle time schedule. Then $L_{\max}(\sigma^{\mathrm{NI}}) \le 2L_{\max}(\sigma^*)$, and no better performance bound exists for an arbitrary no idle time schedule.*

Proof. As in the proof of Theorem 2, (1) holds with σ^{AS} replaced by σ^{NI}. Since $P_j \le C_{\max}(\sigma^*) \le L_{\max}(\sigma^*)$ for a no idle time schedule σ^{NI}, where the final inequality is a consequence of our assumption that $d_j \le 0$ for all jobs j, we obtain $L_{\max}(\sigma^{\mathrm{NI}}) \le 2L_{\max}(\sigma^*)$.

The following instance shows that no better bound exists.

Example 3. $n = n_O = 2$; processing times and due dates are shown in Table 3, where $v \ge 1$; $k = v$; and the intervals within which each job is scheduled in π are also given in Table 3.

Schedules σ^{NI} and σ^* are defined in Table 3, with the values of $\Delta_j(\pi, \sigma^{\mathrm{NI}})$ and $\Delta_j(\pi, \sigma^*)$ for $j \in J_O$ also given. After deducing that $L_{\max}(\sigma^{\mathrm{NI}}) = 2v$ and $L_{\max}(\sigma^*) = v+1$, we obtain $\lim_{v\to\infty}(L_{\max}(\sigma^{\mathrm{NI}})/L_{\max}(\sigma^*)) = \lim_{v\to\infty}(2v/(v+1)) = 2$. □

3.2. A Best Possible Approximation Algorithm

In this section, we propose an approximation algorithm that schedules the jobs of J_O in π order, and then appends the jobs of J_N. Since the appended jobs of J_N are scheduled in EDD order, this algorithm is named *appended earliest due date* (AEDD). We show that this algorithm is best possible among polynomial time approximation algorithms.

AEDD

J_O Construction. Using the sequence π, schedule each job of J_O as early as possible.

J_N Construction. Schedule the jobs of J_N in EDD order after those of J_O, with no idle time between them.

The J_O Construction step requires $O(n_O)$ time, sorting the jobs of J_N in EDD order requires $O(n_N \log n_N)$ time, and scheduling them after J_O requires $O(n_N)$ time. Therefore, the overall time complexity of Algorithm AEDD is $O(n_O + n_N \log n_N)$.

Theorem 5. *Let σ^{AEDD} denote a schedule found by Algorithm AEDD. Then $L_{\max}(\sigma^{\mathrm{AEDD}}) \le 2L_{\max}(\sigma^*)$, and no better performance bound exists for Algorithm AEDD.*

Proof. Let l denote the last job in π (and thus also the last job of J_O in σ^{AEDD}). Since Algorithm AEDD

Table 3 Data for Example 3

Job j	1	2
p_j	1	v
d_j	$-v+1$	0
π	$[v, v+1]$	$[0, v]$
σ^{NI}	$[v, v+1]$	$[0, v]$
$\Delta_j(\pi, \sigma^{\mathrm{NI}})$	0	0
σ^*	$[0, 1]$	$[1, v+1]$
$\Delta_j(\pi, \sigma^*)$	v	1

schedules the jobs of N_O in nondecreasing order of their due dates, the value of $C_l(\sigma^{\mathrm{AEDD}})$ is minimal. Therefore,

$$C_l(\sigma^{\mathrm{AEDD}}) \le C_{\max}(\sigma^*) \le L_{\max}(\sigma^*), \tag{3}$$

where the final inequality is obtained from our assumption that $d_j \le 0$ for all jobs j.

Let job j have maximum lateness in schedule σ^{AEDD}. First, suppose that $j \in J_O$. Using the definition of lateness, we obtain $L_{\max}(\sigma^{\mathrm{AEDD}}) = C_j(\sigma^{\mathrm{AEDD}}) - d_j < L_{\max}(\sigma^*) + C_l(\sigma^{\mathrm{AEDD}})$, where the inequality is a result of job l being last among the jobs in J_O. Substituting (3) yields the desired inequality $L_{\max}(\sigma^{\mathrm{AEDD}}) \le 2L_{\max}(\sigma^*)$.

Alternatively, suppose that $j \in J_N$. In this case, $L_{\max}(\sigma^{\mathrm{AEDD}}) = C_l(\sigma^{\mathrm{AEDD}}) + L_{\max}(J_N)$, where $L_{\max}(J_N)$ is the maximum lateness of an EDD schedule of the jobs of J_N, evaluated by starting processing at time zero. Trivially, $L_{\max}(J_N) \le L_{\max}(\sigma^*)$. Therefore, using (3), we obtain $L_{\max}(\sigma^{\mathrm{AEDD}}) \le 2L_{\max}(\sigma^*)$ in this case also.

The following instance shows that no better bound exists.

Example 4. $n = 2$, $n_O = 1$, $n_N = 1$; processing times and due dates are shown in Table 4, where $v \ge 1$; $k = 1$; and the intervals within which each job is scheduled in π are also given in Table 4.

Schedules σ^{AEDD} and σ^* are defined in Table 4, with the values of $\Delta_j(\pi, \sigma^{\mathrm{AEDD}})$ and $\Delta_j(\pi, \sigma^*)$ for $j \in J_O$ also given. Since $L_{\max}(\sigma^{\mathrm{AEDD}}) = 2v$ and $L_{\max}(\sigma^*) = v + 1$, we obtain $\lim_{v\to\infty}(L_{\max}(\sigma^{\mathrm{AEDD}})/L_{\max}(\sigma^*)) = \lim_{v\to\infty}(2v/(v+1)) = 2$. □

We have thus established that a schedule without idle time (if such a schedule exists) and a schedule provided by Algorithm AEDD have a worst-case performance bound of 2. We now show that 2 is the *best possible* worst-case performance bound of any polynomial time algorithm, under the standard assumption that $d_j \le 0$ for $j \in J$, unless P = NP.

Theorem 6. *Let σ^A denote a schedule delivered by a polynomial time approximation algorithm. If $L_{\max}(\sigma^A) \le \rho L_{\max}(\sigma^*)$ for any problem instance, then $\rho \ge 2$ unless* P = NP.

Proof. By reduction from the following problem, which is known to be NP-complete.

Table 4 Data for Example 4

Job j	1	2
p_j	1	v
d_j	0	0
π	$[v, v+1]$	
σ^{AEDD}	$[v-1, v]$	$[v, 2v]$
$\Delta_j(\pi, \sigma^{\mathrm{AEDD}})$	1	
σ^*	$[v, v+1]$	$[0, v]$
$\Delta_j(\pi, \sigma^*)$	0	

Partition (Garey and Johnson 1979). Given t elements with integer sizes $a_1, \ldots, a_t$, where $\sum_{i=1}^{t} a_i = 2A$, does there exist a partition S_1, S_2 of the index set $\{1, \ldots, t\}$ such that $\sum_{i\in S_1} a_i = \sum_{i\in S_2} a_i = A$?

Given an instance of Partition, we construct an instance of problem $1|\Delta_{\max}(\pi) \le k|L_{\max}$, where $n = t + 2$, $n_O = 2$, $n_N = t$, and $k = 0$. For J_O, $p_1 = 1$, $p_2 = rA$, and $d_1 = d_2 = 0$, where $r \ge 1$ is a constant. For J_N, $p_j = a_{j-2}$ and $d_j = -rA$, for $j = 3, \ldots, t+2$. Schedule π has job 1 in $[A, A+1]$ and job 2 in $[2A+1, 2A+1+rA]$.

Since $k = 0$, jobs 1 and 2 must occupy the same time intervals as in π. Thus, the jobs of J_N are scheduled within $[0, A]$, $[A+1, 2A+1]$ and the time interval starting at $2A+1+rA$.

Suppose that Partition has a solution. Then for each $i \in S_1$, job $i+2$ can be scheduled in $[0, A]$, and for each $i \in S_2$, job $i+2$ can be scheduled in $[A+1, 2A+1]$. This gives $L_{\max} = rA + 2A + 1$, where the maximum lateness is attained both for job 2 and for the last job of J_N.

On the other hand, for any schedule in which at least one job of J_N is scheduled after job 2, the maximum lateness is at least $(2A+1+rA+1) - (-rA) = 2rA + 2A + 2$.

If Partition has a solution but the polynomial time algorithm schedules a job of J_N to complete after job 2, then $L_{\max}(\sigma^H)/L_{\max}(\sigma^*) = (2rA+2A+2)/(rA+2A+1)$, which can be arbitrarily close to 2. Thus, if $\rho < 2$, the polynomial time algorithm schedules all jobs of J_N before job 2, and therefore finds a solution to Partition if such a solution exists. However, this is only possible if P = NP. □

Theorem 6 shows that a polynomial time approximation scheme (Papadimitriou and Steiglitz 1982, Schuurman and Woeginger 2007) is not possible even under the assumption that $d_j \le 0$ for $j \in J$, unless P = NP. This generalizes the result in Corollary 1.

4. Branch and Bound Algorithm

In this section, we present the components of our branch and bound algorithm. Section 4.1 describes our lower bounding scheme that uses a preemptive relaxation. Section 4.2 presents two heuristics for obtaining upper bounds. One of these heuristics provides the starting solution for a variable neighborhood descent procedure that we develop. In Section 4.3, we establish some properties as dominance rules within our branch and bound algorithm. Finally, we present an overview of the complete branch and bound algorithm in Section 4.4.

4.1. Lower Bounds

Our branch and bound algorithm obtains lower bounds on the optimal value of the maximum lateness

using a preemptive relaxation, as follows. Each job j has an implied release date $\bar{r}_j$, and a cost function

$$f_j(C_j) = \begin{cases} C_j - d_j & \text{for } j \in J_N;\ j \in J_O \text{ and } C_j \le \bar{d}_j \\ \infty & \text{for } j \in J_O \text{ and } C_j > \bar{d}_j. \end{cases} \quad (4)$$

Our lower bound is the solution of the preemptive problem $1|pmtn, r_j|f_{\max}$, where $f_{\max} = \max_{j\in J}\{f_j\}$, using the $O(n^2)$ time algorithm of Baker et al. (1983), hereafter referred to as PMTN.

4.2. Upper Bounds

In this section, we present two heuristics for obtaining upper bounds. The first resembles the procedure for computing the lower bound using PMTN. The second computes an initial schedule using a generalization of AEDD, and then improves this solution using a variable neighborhood descent procedure.

The first heuristic, denoted as Adjusted PMTN (APMTN), is motivated by the observation that in most instances very few jobs are preempted by PMTN. For example, our computational results show that when $n = 100$, the number of preemptions generated by PMTN averages 5.6 and does not exceed 26 in any of the 720 instances tested, and when $n = 1{,}000$, the number of preemptions averages 55 and does not exceed 231 (see the experimental design in Section 5). This suggests that a procedure similar to PMTN, but without using preemption, may produce good schedules.

APMTN

Step 0. *Initialization*

Step 0.1. Compute the implied release date $\bar{r}_j = C_j(\pi) - p_j - k$ and the implied deadline $\bar{d}_j = C_j(\pi) + k$ for $j \in J_O$, and set $\bar{r}_j = 0$ and $\bar{d}_j = \infty$ for $j \in J_N$.

Step 0.2. Compute a lower bound using PMTN.

Step 0.3. Schedule the n jobs in nondecreasing order of $\bar{r}_j$, starting each as early as possible. Divide the schedule into blocks, where a block is a maximal period of processing without idle time between the jobs. For each block i, set B_i to be the set of jobs within this block, and compute the maximum lateness of these jobs.

Step 1. *Adjustment*

Step 1.1. Find a block i with the largest maximum lateness of its jobs. If block i has maximum lateness no greater than the lower bound obtained by PMTN or has already been searched, then go to Step 2; otherwise, search block i as follows.

Step 1.2. Find the completion time C_i^b of the last job in B_i, and set $\hat{B}_i = B_i$.

Step 1.3. If $\hat{B}_i = \varnothing$, then terminate with no feasible schedule found; otherwise, find a job $j \in \hat{B}_i$ for which $f_j(C_i^b)$ (as defined in (4)) is minimal. If $C_i^b > \bar{d}_j$, then terminate with no feasible schedule found; otherwise, provisionally schedule job j to be completed at time C_i^b.

Step 1.4. Schedule all of the jobs in $B_i\backslash\{j\}$ in nondecreasing $\bar{r}_j$ order, starting each as early as possible. If idle time occurs between the jobs because of the implied release date values $\bar{r}_j$, then set $\hat{B}_i = \hat{B}_i\backslash\{j\}$, and go to Step 1.3. Otherwise, regard job j as scheduled, and set $B_i = B_i\backslash\{j\}$; if $B_i \neq \varnothing$, then return to Step 1.2; if $B_i = \varnothing$, then return to Step 1.1.

Step 2. *Output*

Output the heuristic schedule and its maximum lateness.

The intuition underlying the APMTN heuristic is as follows. The schedule in Step 1, which minimizes machine idle time, is decomposed into blocks where processing is consecutive. To reduce the overall maximum lateness, a block is chosen for which the maximum lateness of its jobs is greatest, and an attempt is made to reschedule the jobs of this block, without extending its duration. The procedure successively selects a job for the last (unfilled) position in this block, basing decisions on the cost function defined in (4) until the complete block is scheduled. APMTN requires $O(n^3)$ time.

Although our approximation algorithm AEDD achieves the best possible worst-case performance ratio, it does not always find an active schedule or a locally optimal schedule. Therefore, we now describe a modified heuristic, inserted and improved earliest due date (IIEDD), that allows the insertion of the jobs of J_N with those of J_O, and also incorporates a simple local search procedure. Modifications of the classical EDD algorithm (Jackson 1955) appear elsewhere in the scheduling literature; see, for example, Grigoriev et al. (2005).

IIEDD

J_O Construction. Using the sequence π to define the processing order, schedule each job of J_O as early as possible.

J_N Construction. Without delaying any jobs of J_O, consider each job of J_N in turn in EDD order and insert it into the earliest possible position in the partial schedule.

J Improvement. Apply the following for $i = 2, \ldots, n$.

Exchange the jobs in positions $i-1$ and i in the current sequence, if the exchange (a) decreases the maximum lateness of all the jobs, or (b) decreases the maximum lateness of these two jobs without increasing the maximum lateness of all the jobs.

Algorithm IIEDD requires $O(n^2)$ time. While IIEDD is expected on average to be superior to AEDD, even better solutions may be achievable by applying a more sophisticated local search procedure to the solution generated by IIEDD. Specifically, we design

a variable neighborhood descent (VND) procedure, which is described below. A useful introduction to variable neighborhood search procedures is provided by Hansen and Mladenović (2001).

Our VND uses six types of neighborhoods. In the following description of our neighborhood moves, we let B_j and A_j denote the set of jobs that are scheduled before and after job j, respectively, for $j \in J$. Suppose that job j has maximum lateness in the current schedule. In the case of equal maximum lateness values, the job with the larger completion time is selected. In the case where the schedule contains some machine idle time before job j, suppose that job h starts the block containing job j, i.e., job h is immediately preceded by machine idle time and there is no machine idle time between the processing of jobs h and j. To reduce the maximum lateness, job j must be completed earlier, and therefore the set of jobs sequenced between h and j must be reduced. Our neighborhood structures that achieve this reduction are defined as follows:

N_1: remove job j and insert it immediately before some job i, where $i \in B_j$;
N_2: remove some job i and insert it immediately after job j, where $i \in B_j \backslash B_h$;
N_3: swap job i and job j, where $i \in B_j$;
N_4: if h exists, remove job h and insert it immediately after some job i, where $i \in A_h$;
N_5: if h exists, remove some job i and insert it immediately before job h, where $i \in A_h \backslash (A_j \cup \{j\})$;
N_6: if h exists, swap job h and job i, where $i \in A_h \backslash \{j\}$.

For any neighbor, a corresponding schedule is computed by scheduling each job in the sequence as early as possible, subject to the implied release dates.

A neighbor resulting from one of these moves is *preferred* to the previous schedule if the maximum lateness is smaller. First consider N_1, N_2, and N_3. In the case of equal maximum lateness values, the schedule with the earlier completion time of the job in the position previously occupied by job j is preferred. If there is still a tie, then the neighbor is preferred to the previous schedule if $d_j < d_i$ or if $d_j = d_i$ and $j < i$. Now consider N_4, N_5, and N_6. In the case of equal maximum lateness values, the schedule with the earlier completion time of the job in the position previously occupied by job i is preferred. If there is still a tie, then the neighbor is preferred to the previous schedule if $\bar{r}_i < \bar{r}_h$ or if $\bar{r}_i = \bar{r}_h$ and $i < h$.

VND

Initialization. Apply IIEDD to find an initial schedule σ.

Search Neighborhoods. Execute the following for $l = 1, \ldots, 6$.

Apply the following procedure until none of the neighbors of N_l of the current schedule σ is preferred to σ.

Find a preferred neighbor σ' of σ with the smallest maximum lateness, and set $\sigma = \sigma'$.

Termination Test. If σ has changed during the previous application of the Search Neighborhoods step, then return to the Search Neighborhoods step; otherwise, stop.

The choice of IIEDD to provide initial solutions for VND is suggested by our preliminary computational tests. More discussion can be found in Section 5.1.

4.3. Structural Properties

In this section, we describe several structural properties that are used as dominance rules to eliminate nodes of the search tree in our branch and bound algorithm.

PROPERTY 1. *For any two jobs i and j, where $i, j \in J_O$, if $C_i(\pi) < C_j(\pi)$ and $C_j(\pi) - C_i(\pi) > 2k - p_i$, then in any feasible schedule job i precedes job j.*

PROOF. Consider a feasible schedule σ in which job j precedes job i so that $C_j(\sigma) \le C_i(\sigma) - p_i$. Feasibility ensures that $C_j(\sigma) \ge \bar{r}_j + p_j = C_j(\pi) - k$ and $C_i(\sigma) \le \bar{d}_i = C_i(\pi) + k$. From these inequalities, $C_j(\pi) - k \le C_j(\sigma) \le C_i(\sigma) - p_i \le C_i(\pi) + k - p_i$, which implies that $C_j(\pi) - C_i(\pi) \le 2k - p_i$, thus contradicting the condition of the property. □

Property 1 establishes the order of certain pairs of jobs before applying branch and bound. A node of the branch and bound search tree created by our branching rule corresponds to a partial sequence of jobs $\sigma = (\sigma(1), \ldots, \sigma(l))$, where these jobs are each scheduled in turn as early as possible. Property 1 may allow nodes of the search tree to be discarded.

A partial sequence σ is *lexicographically smaller* than another partial sequence σ' if there exists some position j in the partial sequences such that $\sigma(i) = \sigma'(i)$ for $i = 1, \ldots, j-1$ and $\sigma(j) < \sigma'(j)$.

PROPERTY 2. *For a partial schedule $\sigma = (\sigma(1), \ldots, \sigma(l))$, suppose that a candidate job k is scheduled as early as possible immediately after job $\sigma(l)$. Let* LB *denote a lower bound on all schedules that start with σ, and let* UB *denote the value of $L_{\max}$ in the best known feasible schedule. If one or more of the following five conditions is satisfied, then the search for a schedule with $L_{\max} <$ UB can be limited by not sequencing job k after σ in position $l+1$.*

1. *The lateness of job k in position $l+1$ is greater than or equal to* UB.

2. *After sorting all the unscheduled jobs of J_O in nondecreasing order of their implied deadlines, and scheduling them in this order as early as possible after job k which is in position $l+1$, there exists a job with completion time greater than its implied deadline.*

3. *Jobs $\sigma(1), \ldots, \sigma(l)$, k, can be permuted into a sequence with smaller maximum completion time, and keeping their maximum lateness less than or equal to* LB.

4. *Jobs* $\sigma(1), \ldots, \sigma(l)$, k, *which have no idle time between them in* σ, *can be permuted into a lexicographically smaller sequence, keeping their maximum completion time unchanged, and keeping their maximum lateness less than or equal to* LB.

Property 2 is a composition of several properties that are intuitive; therefore we omit the proof. The four conditions in this property are used as dominance rules to eliminate redundant nodes within the branch and bound algorithm. Conditions 1 and 2 are based on lateness and feasibility considerations. The nodes eliminated by the initial tests defined in Conditions 1 and 2 are classified as eliminated by INI in Tables 7 and 8 in Section 5.2.

In Conditions 3 and 4, there is an exponential number of sequences to be considered. Therefore, we consider the permutation of only the last four jobs in σ together with job k, when there is no idle time between them under Condition 4. If there are fewer than four jobs before k, then we consider all the jobs in σ. The nodes eliminated by Conditions 3 and 4 are mainly classified as eliminated by PMU in Tables 7 and 8 in Section 5.2. However, we distinguish the commonly used single exchange of jobs $\sigma(l)$ and k from the other cases, by classifying the nodes eliminated by this exchange as eliminated by INI in Tables 7 and 8.

Conditions 3 and 4 extend ideas of Potts and Van Wassenhove (1983), Posner (1985), and Pan (2003). The first two papers apply these ideas for the two most recently fixed jobs in the partial schedule. By applying them for up to 15 fixed jobs in the partial schedule, Pan achieves better performance. However, when several nodes exist with the same best objective value, Pan does not eliminate any of them, even though considering a single one is enough. We use lexicographical ordering in Condition 4 to strengthen the node elimination rule.

Finally, the nodes eliminated by standard lower bounding considerations are classified as eliminated by LB in Tables 7 and 8 in Section 5.2.

4.4. Branch and Bound Strategy

Various procedures are used at the root node of the search tree in our branch and bound algorithm. First, a lower bound is computed using PMTN. If no job is preempted in this procedure, then an optimal solution has been found and the algorithm terminates. Otherwise, we apply APMTN, IIEDD, and VND in succession, terminating the algorithm immediately if a solution is found with maximum lateness equal to the lower bound. The order of applying these procedures is justified by our computational results in Section 5.1. The best upper bound found at the root node is computed.

As indicated, our branch and bound algorithm uses a classical forward branching strategy for which a node at level l of the search tree corresponds to an initial partial sequence $(\sigma(1), \ldots, \sigma(l))$ in which jobs are fixed in the first l positions. The tree evolves by branching from a currently active node. The candidate list of jobs to be scheduled next is reduced using Property 1 as a dominance rule. Wherever possible, nodes generated by the remaining candidates are eliminated using Property 2.

For each node that is not eliminated by Properties 1 and 2, a lower bound is computed using PMTN. Then APMTN and IIEDD are used, and the better of these two upper bounds is selected. If this upper bound is equal to the overall lower bound, then an optimal solution is achieved, and we terminate the algorithm. Otherwise, if the upper and lower bounds for the new node are equal, then we eliminate this node and update the overall upper bound if necessary. If all candidate jobs for position $l+1$ are eliminated, then we backtrack.

A computation time limit of 60 seconds is imposed, at which stage the branch and bound algorithm terminates even if an optimal solution has not been found.

5. Computational Results

Our data set consists of 7,200 problem instances, randomly generated as follows. Using the guidelines established by Hall and Posner (2001), (a) we generate a wide range of parameter specifications, (b) all the parameters can be rescaled together without significantly affecting performance, and (c) the experimental design varies only the parameters that may affect the analysis. We first select $n \in \{20, 40, 60, 80, 100, 200, 400, 600, 800, 1000\}$ and $n_O \in \{0.25n, 0.5n, 0.75n\}$, and generate $p_j \sim U[1, \ldots, p^{\mathrm{UB}}]$ where $p^{\mathrm{UB}} \in \{20, 50\}$, and $d_j \sim U[0, \ldots, d^{\mathrm{UB}}]$ where $d^{\mathrm{UB}} = (1 + p^{\mathrm{UB}})n/2 - 1$. Assuming that the jobs are indexed so that $J_O = \{1, \ldots, n_O\}$, we generate π by first sequencing these jobs in the order $(1, \ldots, n_O)$, and then inserting idle time periods. The lengths of the idle time periods follow the same distribution as the processing times of the jobs. Let FIT denote the frequency of idle time within the original schedule; for example, if $\mathrm{FIT} = 0.1$ and $n_O = 40$, then the expected number of idle time periods is $0.1 \times 40 = 4$. We consider $\mathrm{FIT} \in \{0.1, 0.5, 0.9\}$. The value of k is determined by a parameter RK that models a wide range of tightness for the maximum time disruption constraint and is a multiple of the expected makespan of the original schedule. The expected makespan is obtained from the average number of intervals of processing or idle time, $(n_O + \mathrm{FIT}n_O)$, and the average length of these intervals, $(1 + p^{\mathrm{UB}})/2$. Thus, we set $k = \mathrm{RK}((n_O + \mathrm{FIT}n_O)(1 + p^{\mathrm{UB}})/2)$, where $\mathrm{RK} \in \{0.1, 0.3, 0.5, 0.7\}$.

Table 5 Effect of n on Performance of the Heuristics

	PMTN	APMTN			IIEDD			VND						
n	NO	NO	AGP	MGP	NO	AGP	MGP	NO	AGP	MGP	ANI	MNI	AT	MT
20	239	513	5.9	43	340	35.9	256	625	8.1	35	1.42	18	0.00	0.02
40	173	483	4.3	23	277	85.3	667	620	9.9	53	4.40	48	0.00	0.47
60	139	512	5.2	390	217	117.7	875	638	12.1	149	7.37	81	0.01	0.05
80	113	500	3.0	27	176	163.5	1,289	617	11.4	108	11.32	108	0.01	0.11
100	92	519	2.7	21	177	197.2	1,531	618	16.1	177	14.63	120	0.02	0.16
200	36	593	1.9	12	160	369.2	2,802	619	43.6	550	28.27	297	0.08	0.78
400	21	665	1.5	5	155	738.6	5,482	622	99.1	630	56.87	530	0.35	3.50
600	6	686	1.2	3	145	1,090.5	8,792	586	155.7	1,056	85.79	871	0.92	9.67
800	5	703	1.5	3	150	1,284.6	10,765	624	237.7	2,606	86.76	881	1.52	17.75
1,000	1	713	1.1	2	148	1,701.1	12,626	563	298.8	2,626	124.39	1,267	2.99	31.08

For each value of n and for each of the 72 possible combinations of the other parameters described above, we randomly generate 10 problem instances. The algorithm is coded in Microsoft Visual C++, and run on a 3.2 GHz Pentium IV personal computer with 1 GB of memory. Section 5.1 provides a comparison of our heuristics on the data set described above. In Section 5.2, we test our branch and bound algorithm first on the data set described above, on an additional data set consisting of 200 difficult problem instances, and finally on 1,000 instances specifically generated by multiple job arrivals. Sensitivity to the due dates is also considered.

5.1. Heuristics

This section provides a comparison of our heuristics within the branch and bound algorithm. We compare the value of each heuristic solution with the optimal solution value that is obtained using branch and bound. In Table 5 and the other tables that follow, let NO denote the number of instances for which an optimal solution is obtained. Where an optimal solution is not obtained, AGP and MGP represent the average and maximum gap, respectively, where the gap is the difference between the value of a heuristic solution and the optimal solution value. We note that PMTN finds a lower bound but not necessarily a feasible solution. Therefore, NO counts the number of instances for which PMTN finds a feasible, and therefore optimal, solution. For both APMTN and IIEDD, we provide NO, AGP, and MGP. It is possible that AMPTN does not find a feasible schedule. However, in our computational study, APMTN always finds a feasible schedule except for six instances with $n = 20$. For VND, we provide NO, AGP, MGP, ANI, MNI, AT, and MT, where ANI and MNI are the average and maximum number of times that VND improves upon the initial schedule, and AT and MT are the average and maximum computation times in seconds. To obtain consistent results, we apply VND even for instances where IIEDD generates an optimal solution.

Table 5 shows the results of our computational comparison of heuristics. PMTN is effective at finding feasible, and hence optimal, solutions for smaller problems. Also, APMTN and VND routinely find solutions that are very close to optimal for problems of all sizes. However, whereas the performance of APMTN improves as n increases, VND become less effective. Furthermore, IIEDD is not effective at finding good solutions, but it quickly finds solutions that VND is capable of improving substantially. Using solutions from APMTN as a starting point for VND is not effective, since those solutions are often locally optimal. Our computational results show that increasing either n_O/n or p^{UB}, or decreasing FIT, makes the performance of all the heuristics less reliable. Also, our results show that an increase in RK is associated with poorer performance of IIEDD.

Our computational results suggest that the best order in which to apply the heuristics is PMTN, APMTN, IIEDD, VND. We start with PMTN because it is the only procedure that can verify an optimal solution. APMTN follows next because it delivers very good solutions that can be verified by the lower bound from PMTN. IIEDD is used after that because it provides solutions that are often excellent starting points for VND.

5.2. Branch and Bound Algorithm

Tables 6 and 7 summarize our computational results for the 7,200 randomly generated problem instances. Where shown in those tables, PMTN, APMTN, IIEDD,

Table 6 Effect of n on Performance of the Algorithm

n	PMTN	APMTN	IIEDD	VND	BB	UNS	AT	MT
20	239	263	65	84	69	0	0.01	1.34
40	173	306	58	117	66	0	0.02	7.66
60	139	373	59	110	36	3	0.01	4.47
80	113	387	49	126	44	1	0.01	4.70
100	92	427	46	113	40	2	0.08	30.03
200	36	557	13	85	28	1	0.11	40.98
400	21	644	6	40	9	0	0.04	2.03
600	6	680	4	21	9	0	0.08	9.33
800	5	698	3	12	2	0	0.06	12.34
1,000	1	712	1	4	2	0	0.12	24.23

Table 7 Detailed Results for the Branch and Bound Algorithm

n	BB	AN	MN	PMTN	APMTN	IIEDD	ENU	INI	PMU	LB
20	69	139	3,366	31	9	9	20	1,278	39	231
40	66	285	9,663	42	15	3	6	4,246	102	1,448
60	36	165	86,383	25	10	1	0	5,394	0	135
80	44	75	52,711	25	17	2	0	3,166	1	217
100	40	672	51,547	19	19	1	1	39,806	0	2,211
200	28	1,192	28,238	11	17	0	0	73,823	0	2,400
400	9	102	191	2	7	0	0	1	0	0
600	9	130	277	1	7	1	0	1	0	0
800	2	159	246	0	1	1	0	1	0	0
1,000	2	109	146	0	1	1	0	0	0	0

VND, and BB denote the number of instances with optimal solutions found by PMTN, the APMTN heuristic, the IIEDD heuristic, the VND procedure, and the branch and bound algorithm, respectively. In Table 6, these numbers refer to the part of the branch and bound algorithm where termination occurs. Also, UNS denotes the number of instances that remain unsolved after 60 seconds of CPU time. Finally, AT and MT are as defined in Table 5.

In most instances, the difference between the maximum lateness of an optimal schedule and the lower bound is small. Among the 7,200 instances, there are only 28 with optimal solution value strictly greater than the lower bound. Over these 28 instances, the mean difference between the optimal solution value and the lower bound is 3.07, and the maximum difference is 10. Also, the 25th, 50th, and 75th percentiles for the difference are 1, 2, and 4, respectively. Moreover, when $n \geq 200$, all the solved instances have optimal solution values equal to their preemptive lower bounds. Even for such instances, finding a feasible and optimal nonpreemptive solution may still require substantial computation.

Also, PMTN, APMTN, and IIEDD are computationally successful at finding optimal schedules, as verified by a lower bound from PMTN. One reason is that maximum lateness is a bottleneck measure, which may be attained for only a few jobs, so that resequencing several jobs may leave the maximum lateness unaltered. Especially when $n \geq 60$, there are often many optimal schedules. Therefore, PMTN, APMTN, and IIEDD can find an optimal schedule in many instances. A second reason is that problem $1|\Delta_{\max}(\pi) \leq k|L_{\max}$ is closely related to the classical scheduling problem $1|r_j|L_{\max}$. Among intractable problems, the latter is known to be "relatively easy" (Hall and Shmoys 1992). Moreover, the specially structured implied deadlines in problem $1|\Delta_{\max}(\pi) \leq k|L_{\max}$ may reduce the number of feasible solutions.

From Table 6, we can see that the most difficult problem instances, including all those that remain unsolved, have between 60 and 200 jobs. This is because, as the number of jobs increases, APMTN becomes more powerful. We also note from Tables 5 and 6 that PMTN and IIEDD gradually become less effective as the number of jobs increases. For the solved instances, the relationship between n and AT is not monotone. Overall, the results in Table 6 show that our branch and bound algorithm quickly solves 7,193 out of 7,200 instances of this unary NP-hard problem with up to 1,000 jobs.

Parameter values that yield more optimal schedules tend to produce easier problems. This is observed when n_O/n decreases, where more jobs in J_O leads to a loss of flexibility in scheduling the jobs, and when P^{UB} is smaller.

The value of RK has differing effects on various procedures. Smaller RK values make IIEDD more effective because of the similarity between the π and σ^* schedules when implied release date and deadline constraints are tight. However, the tightness of these constraints also leads to many preemptions in PMTN. Hence, for small RK values, PMTN is less effective at finding feasible nonpreeemptive schedules, and APMTN is similarly less effective.

In Table 7, AN and MN denote the average and maximum number of nodes that are not directly eliminated by the dominance rules, permutation procedure, or lower bound in the solved instances (rounded to the nearest integer). We provide a breakdown of the contributions of various procedures within branch and bound; PMTN, APMTN, IIEDD, and ENU denote the numbers of instances with optimal schedules that are found by PMTN, APMTN, IIEDD, and implicit enumeration within the branch and bound algorithm, respectively. Also within branch and bound, INI, PMU, and LB denote the total numbers of nodes eliminated by the initial tests in the dominance rules, by permutation of the last five jobs of the partial schedules, and by lower bounds for the partial schedules, respectively.

From Table 7, when $n \leq 200$, the average number of nodes examined increases with the number of jobs, up to about 1,200 nodes. However, for instances with $400 \leq n \leq 1{,}000$, the branch and bound algorithm consistently examines between 100 and 200 nodes. Within the branch and bound algorithm, one of the heuristics frequently finds a schedule with value equal to the initial lower bound, leaving only a few instances for which an optimal schedule is generated by branch and bound. Also, the dominance rules INI eliminate the most nodes, the lower bound procedure using PMTN is the next most successful, and the permutation of the last five jobs at the current node also eliminates some nodes for small problem instances. For instances with $n \geq 400$, these three elimination methods are used only rarely, since the total number of nodes examined is small. Moreover, in almost every case, PMTN, APMTN, or IIEDD finds an optimal schedule with value equal to the global lower bound directly from a partial schedule, without using the three elimination

Table 8 Effect of n on Performance of the Algorithm on Difficult Instances

n	AG	MG	AT	MT	SUB	UNS	TN	AN	MN	INI	PMU	LB
20	7.4	21	0.04	1.03	36	0	325	119	3,397	1,086	13	197
40	7.9	34	4.03	32.64	39	0	1,471	4,156	31,876	101,259	3,909	16,348
60	8.1	32	59.29	690.56	33	3	2,957	33,156	336,044	1,011,440	73,482	264,730
80	5.9	27	184.28	1,149.39	28	17	7,766	83,092	691,309	3,213,048	182,574	569,518
100	4.0	16	324.13	1,199.94	28	92	24,878	81,618	322,991	4,530,422	173,208	1,169,770

methods. For the seven unsolved instances, the absolute gap values are very small: 1, 1, 1, 2, 2, 2, and 6. Moreover, all these seven instances are solved optimally by branch and bound within 23 minutes.

To evaluate the performance of the branch and bound algorithm on more difficult problems, we construct a data set of 200 problem instances with nonzero gaps between the optimal solution value and the lower bound found by PMTN. In Tables 6 and 7, such instances are rarely found for $n > 100$. Therefore, we test $n \in \{20, 40, 60, 80, 100\}$. For the other parameters, we set $n_O/n = 0.25$, $p^{\mathrm{UB}} = 50$, FIT $= 0.1$, and RK $= 0.1$. We generate problem instances until we have found 40 for each value of n that (a) have nonzero gaps and (b) are solved by the branch and bound algorithm within 20 minutes. Our results are summarized in Table 8, where AG and MG are the average and maximum value of gaps. Next, SUB denotes the number of instances out of 40 with optimal schedules found by heuristics at the initial node of the branch tree. Also, UNS denotes the number of instances with nonzero gaps that are not solved within the 20 minute time limit; for example, 92 of the first 132 instances generated with nonzero gaps for $n = 100$ are unsolved after 20 minutes. Finally, TN denotes the total number of instances that need to be generated to obtain 40 instances satisfying properties (a) and (b) above for each value of n considered.

Table 8 shows that when there is a gap between the optimal solution value and the lower bound, the branch and bound algorithm typically requires more time than for other instances. However, such instances occur only very rarely except when $n \le 40$, as shown in column TN. Moreover, column SUB indicates that the heuristics find an optimal schedule at the initial node of the branch tree in 164 out of the 200 difficult-to-solve instances. Finally, columns AN, MN, INI, PMU, and LB are defined as for Table 7. The results in Table 8 are similar to those in Table 7, except that many more nodes are examined here.

Next, we consider *ancestor instances*, a type of problem that directly models the motivation for the rescheduling problem discussed in Section 1. Suppose that there is an initial EDD sequence of jobs without idle time between them, and then multiple sets of new jobs arrive. We assume that, between consecutive arrivals of two new jobs, some previously available jobs complete their processing. More specifically, we initially assume that n_O jobs have been scheduled but not yet processed, and then n_N new jobs arrive. After the rescheduling of these $n_O + n_N$ jobs, n_N jobs are processed, and then a new set of n_N jobs arrives, and so on. Compared with other instances, ancestor instances are more likely to have previously available jobs that closely follow an EDD sequence.

In Table 9, we show results for ancestor instances with $n_O = 200$ and $n_N = 80$, and where 100 instances are tested. For all these instances, $p^{\mathrm{UB}} = 50$, and $k = 0.1((1 + 50)n/2)$, where $n = n_O + n_N = 280$. In the first column, G denotes the generation number of the problem instance, and the other columns have the same meanings as those in Table 6. Thus, if $G = 1$, the original n_O jobs are sequenced in EDD order, but as G increases the sequence of the jobs moves gradually away from EDD.

From Table 9, most ancestor instances are solved by PMTN or APMTN, and are thus easy to solve, particularly when $n_N \le 0.3n_O$.

We also investigate the effect of clustering of due dates on the performance of the algorithm. Specifically, we let each job have an integer due date that is randomly generated by one of the ten normal distributions with mean $(2i + 1)d^{\mathrm{UB}}/20$ and standard deviation $d^{\mathrm{UB}}/60$, where $i = 0, \ldots, 9$. Because jobs tend to have similar due dates in these instances, there are often multiple optimal schedules and it is easier to find one of them. Consequently, our computational results improve on those reported in Tables 6 and 7.

We also conduct a second experiment involving the distribution of the due dates. We use a triangular distribution to cluster the due dates towards the front of the planning horizon. This models many practical situations where a manufacturer has made due date commitments to more customers in the near future

Table 9 Effect of Generation Number on Performance for Ancestor Instances

G	PMTN	APMTN	IIEDD	VND	BB	UNS	AT	MT
1	100	0	0	0	0	0	0.00	0.02
2	3	85	0	9	3	0	0.12	3.33
3	25	73	0	2	0	0	0.01	0.05
4	33	60	0	2	4	1	0.05	1.84
5	21	73	0	1	5	0	0.05	2.48
6	23	68	1	2	3	3	0.07	2.92
7	17	74	0	5	3	1	0.05	1.99
8	23	72	0	3	2	0	0.03	1.72
9	20	73	0	5	1	1	0.02	1.14
10	13	84	0	1	2	0	0.02	1.06

than in the distant future. Our computational results here also improve on those reported in Tables 6 and 7.

6. Concluding Remarks

In this paper we consider the problem of rescheduling to minimize the maximum lateness after the unexpected arrival of new jobs, while controlling the amount of disruption. Our model analyzes responses to repeated arrivals of new jobs. The effect of disruption is measured by the maximum change in completion time, for any of the original jobs. We describe several classes of schedules for this intractable problem, and analyze their worst-case performance. We design an approximation algorithm and show that it is best possible. Further, we propose a branch and bound algorithm that solves 7,193 out of 7,200 instances tested with up to 1,000 jobs within 60 seconds of computation time.

This work is important because it describes the first optimal algorithm for an intractable rescheduling problem. More generally, it demonstrates that large instances of a unary NP-hard rescheduling problem can be solved optimally. To our knowledge, very few unary NP-hard scheduling problems with instances that are generated following the guidelines in the scheduling literature are solvable to this extent. Apparently, the only other work that reports similar success is by Carlier (1982).

There are several important topics for future research. First, our branch and bound algorithm can be adapted to solve problem $1|\text{prec}, r_j|f_{\max}$. Second, other rescheduling problems need to be studied by designing optimal algorithms. Third, different measures of both schedule cost and disruption cost deserve further investigation. Fourth, problems with several disruptions that occur simultaneously but cannot be modeled as a single disruption need research attention. Finally, our study of single machine rescheduling provides insights into rescheduling issues that can be used in more general manufacturing environments.

Acknowledgments

This research was supported in part by the National Science Foundation (Grants DMI-9821033 and DMI-0421823), the Summer Fellowship Program (Fisher College of Business, The Ohio State University), the Engineering and Physical Sciences Research Council (UK) (Grant GR/R97696/01), and by INTAS (Grant 03-51-5501).

References

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. *Eur. J. Oper. Res.* **161** 86–110.

Baker, K. R., E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan. 1983. Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Oper. Res.* **31** 381–386.

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.

Brucker, P. 1998. *Scheduling Algorithms*, 2nd ed. Springer, Berlin, Germany.

Carlier, J. 1982. The one-machine sequencing problem. *Eur. J. Oper. Res.* **11** 42–47.

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.

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

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.

Grigoriev, A., M. Holthuijsen, J. van de Klundert. 2005. Basic scheduling problems with raw materials constraints. *Naval Res. Logist.* **52** 527–535.

Hall, L. A., D. B. Shmoys. 1992. Jackson's rule for single-machine scheduling: Making a good heuristic better. *Math. Oper. Res.* **17** 22–35.

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., C. N. Potts. 2007. Rescheduling for job unavailability. *Oper. Res.* Forthcoming.

Hansen, P., N. Mladenović. 2001. Variable neighborhood search: Principles and applications. *Eur. J. Oper. Res.* **130** 449–467.

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

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.

Pan, Y. P. 2003. An improved branch and bound algorithm for single machine scheduling with deadlines to minimize total weighted completion time. *Oper. Res. Lett.* **31** 492–496.

Papadimitriou, C. H., K. Steiglitz. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ.

Posner, M. E. 1985. Minimizing weighted completion times with deadlines. *Oper. Res.* **33** 562–574.

Potts, C. N., L. N. Van Wassenhove. 1983. Algorithm for single machine sequencing with deadlines to minimize total weighted completion time. *Eur. J. Oper. Res.* **12** 379–387.

Schuurman, P., G. J. Woeginger. 2007. Approximation schemes—A tutorial. R. H. Möhring, C. N. Potts, A. S. Schulz, G. J. Woeginger, L. A. Wolsey, eds. *Lectures in Scheduling*. Forthcoming.

Szelke, E., R. M. Kerr. 1994. Knowledge-based reactive scheduling. *Production Planning Control* **5** 124–145.

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. 1993. One-machine rescheduling heuristics with efficiency and stability as criteria. *Comput. Oper. Res.* **20** 1–14.

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.