https://doi.org/10.1016/j.dam.2016.03.018

Multitasking via Alternate and Shared Processing: Algorithms and Complexity

Nicholas G. Hall *

Joseph Y.-T. Leung †

Chung-Lun Li ‡

* Fisher College of Business, The Ohio State University, Columbus, Ohio 43210; hall.33@osu.edu

† Department of Computer Science, New Jersey Institute of Technology, Newark, New Jersey 07102; leung@cis.njit.edu

‡ Department of Logistics and Maritime Studies, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong; chung-lun.li@polyu.edu.hk; corresponding author

September 30, 2014

Revised November 6, 2015

Revised February 20, 2016

Revised March 17, 2016

Abstract

This work is motivated by disruptions that occur when jobs are processed by humans, rather than by machines. For example, humans may become tired, bored, or distracted. This paper presents two scheduling models with multitasking features. These models aim to mitigate the loss of productivity in such situations. The first model applies "alternate period processing" and aims either to allow workers to take breaks or to increase workers' job variety. The second model applies "shared processing" and aims to allow workers to share a fixed portion of their processing capacities between their primary tasks and routine activities. For each model, we consider four of the most widely studied and practical classical scheduling objectives. Our purpose is to study the complexity of the resulting scheduling problems. For some problems, we describe a fast optimal algorithm, whereas for other problems an intractability result suggests the probable nonexistence of such an algorithm.

Keywords: Scheduling, motivations for multitasking, efficient algorithm, intractability.

1 Introduction

When jobs are processed by humans rather than machines, different issues arise. For example, humans may become tired, bored, or distracted. These issues disrupt work and often result in a significant loss of productivity. Hence, companies look for system designs that can alleviate this loss. In this paper, we study a simple scheduling system that faces such disruptions, and we propose and analyze two system designs for two different types of disruptions.

The first type of disruption involves workers becoming tired by long work hours or bored by the repetitive nature of their work. As a response, we divide the time horizon into alternating work periods of lengths τ_o and τ_e. We refer to the work periods of length τ_o as *odd periods* and to the work periods of length τ_e as *even periods*. We require each task to be processed either completely within the odd periods or completely within the even periods. For example, a task that is partly processed in period 1 cannot be further processed in period 2 or any other even periods; however, it can be processed further in period 3 or any other odd periods. An application of this design is to operate a service center in two shifts, e.g., a day shift and a night shift, where one worker (or one team of workers) serves the first shift, and another worker (or team) serves the second shift. Each task can only be served by one worker (or team), and therefore must stay within either shift. This system design offers advantages where confidentiality or personal service considerations are important. Example applications include tax preparation, financial audit, legal, medical, counseling, and other professional services. Further, operating the service center in two shifts enables the company to increase the utilization of office space and to provide longer service hours without disrupting the operation or overloading the workers. In this application, the service center is viewed as a processor, and the workers' (or teams') availability imposes a constraint on the job schedule. Another application allows a worker to divide his/her workday into two periods and work on two different sets of tasks at two different locations (e.g., a work office and a home office) during the two periods, thus providing greater personal convenience and job variety. In this application, the worker is viewed as a processor, and the office location imposes a constraint on the job schedule.

The second type of disruption involves workers becoming distracted, for example, by the need to undertake routine tasks such as system maintenance or intraoffice communications.

In response, we propose a system design that allows a worker (or team) to continue work on its main, or *primary* tasks. However, it allocates a fixed percentage, say $100 \times (1 - e)\%$, of its processing capacity to process routine scheduled activities as they occur. This allows the routine scheduled activities, for example, administrative meetings, maintenance work, or meal breaks, to be completed promptly without stopping the primary tasks. An application of this design designates a two-hour lunch period per day, letting one half of the working team have a one hour lunch break during each hour. In this example, the working team is viewed as a processor, and $e = 0.5$. One advantage of this arrangement is to keep the office open continuously so that no incoming request is missed. Another application allows rotating Saturday shifts, where the company only maintains a fraction, for example 33%, of the workforce every Saturday.

Both of these designs make use of the concept of multitasking, even though they are developed for two very different motivations. The first design allows the processor to put an unfinished task on hold and switch to another task, while the second design allows the processor to process two tasks simultaneously with shared capacity. These designs are developed for application to a wide variety of situations. We anticipate that they could be used as simple workplace rules which workers would be expected to follow. They are not intended to be robust against all possible instances.

The concept of multitasking is a familiar one in computer systems. It can be defined as follows, "In computing, multitasking is a method where multiple tasks, also known as processes, share common processing resources such as a CPU" [1]. An operational definition of multitasking is that more than one task can be partially executed at the same time. Sachdeva and Panwar [19] review various scheduling algorithms that are needed for multitasking. Multitasking algorithms for generic applications are discussed by [7]. Models and algorithms for efficient computer multitasking in specific applications have been studied by various researchers. These include Noguera and Badia [17] and Steiger et al. [21] for reconfigurable architectures, and Wang et al. [24] for energy-sensitive dynamic slack allocation.

Scheduling systems with human multitasking are studied by Hall et al. [8]. They identify several principal motivations for multitasking, and provide supporting references from the literature of behavioral psychology, operations management, cognitive engineering, and

project management. Those motivations include:

(i) a need to feel or appear productive;

(ii) a need to demonstrate progress on different tasks or treat task owners equitably;

(iii) anxiety about the processing requirements of waiting tasks;

(iv) a need for variety in work; and

(v) interruption by routine scheduled activities.

Since classical scheduling algorithms typically fail to achieve optimal efficiency in the presence of multitasking, the authors develop new solution procedures for those problems. They also study the extent by which multitasking increases scheduling cost or value. However, the models developed in that work do not distinguish between different motivations for multitasking. Hence, when a primary task is being processed, it is interrupted by all the unfinished tasks. By contrast, our first system design can be viewed as a possible solution to motivation (iv), while our second system design can be viewed as a possible solution to motivation (v).

Our first design is similar to the classical two-parallel-machine scheduling problem, however the first machine is unavailable during periods $[\tau_o, \tau_o + \tau_e], [2\tau_o + \tau_e, 2(\tau_o + \tau_e)], [2(\tau_o + \tau_e) + \tau_o, 3(\tau_o + \tau_e)], \ldots$, and the second machine is unavailable during periods $[0, \tau_o], [\tau_o + \tau_e, 2\tau_o + \tau_e], [2(\tau_o + \tau_e), 2(\tau_o + \tau_e) + \tau_o], \ldots$ Our second design is related to scheduling a single machine with machine unavailability. For example, if $e = 0$, the worker (or team) becomes unavailable whenever he/she encounters a scheduled routine activity. Hence, both designs are scheduling models with some machine unavailability. Ma et al. [15] provide a survey of 85 papers on scheduling with deterministic machine unavailability periods. Kaabi and Harrath [13] provide a survey of parallel machine scheduling with machine availability constraints. More recent works within this research stream include [2, 9, 11, 14, 25], which consider various single and parallel machine models with machine unavailability periods.

The scheduling environment that we consider, under both system designs, is as follows. Consistent with the classical scheduling terminology, we refer to the work center as a "machine" and the tasks as "jobs." We let $N = \{1, \ldots, n\}$ denote a given set of jobs with integer data. Job j has a processing requirement $p_j > 0$, and we let $P = \sum_{j=1}^{n} p_j$. If job j has to share processing capacity with other jobs, the time during which it is being processed may exceed p_j. In some of the problems we consider, job j also has a due date $d_j \geq 0$ and/or a

weight $w_j > 0$, for $j = 1, \ldots, n$. A single machine is available for processing the jobs.

In any feasible schedule σ, we let $C_j(\sigma)$ denote the completion time of job j. We define the lateness of job j as $L_j(\sigma) = C_j(\sigma) - d_j$, and we let $L_{\max} = \max_{1 \le j \le n}\{L_j\}$, and $U_j(\sigma) = 1$ if $C_j(\sigma) > d_j$ and $U_j(\sigma) = 0$ otherwise. Whenever the schedule being considered is clear from context, we omit the argument σ.

We consider the minimization of four objective functions: the total weighted completion time $\sum_{j=1}^{n} w_j C_j$, the total completion time $\sum_{j=1}^{n} C_j$, the maximum lateness $L_{\max} = \max_{1 \le j \le n}\{L_j\}$, and the number of late jobs $\sum_{j=1}^{n} U_j$. These objectives are among the most widely studied and best practically motivated objectives within the scheduling literature [18]. Using the three-field $\alpha \mid \beta \mid \gamma$ notation introduced by Graham et al. [6], the classical versions of these scheduling problems are denoted by $1 \mid\mid \sum w_j C_j$, $1 \mid\mid \sum C_j$, $1 \mid\mid L_{\max}$, and $1 \mid\mid \sum U_j$, respectively. We use "*alt*" in the β field to denote our first system design, and use "$share(e)$" in the β field to denote our second system design.

The following indexing rules from classical scheduling theory are useful in our work. A shortest weighted processing time (SWPT) sequence schedules the jobs in index order, where $w_1/p_1 \ge \cdots \ge w_n/p_n$ [20]. A shortest processing time (SPT) sequence schedules the jobs in index order, where $p_1 \le \cdots \le p_n$ [20]. An earliest due date (EDD) sequence schedules the jobs in index order, where $d_1 \le \cdots \le d_n$ [12].

The remainder of the paper is organized as follows. Sections 2 and 3 consider the two system designs described above. Within each section, we discuss the solvability of scheduling problems with the four objectives defined above. Section 4 provides a conclusion and some suggestions for future research.

2 Alternate Period Processing

As discussed in Section 1, our first system design divides the time horizon into work periods of lengths τ_o and τ_e, where the two work periods alternate. We require each job to be processed completely in the odd periods or completely in the even periods. This constraint is implemented by the construction of odd-period schedules in periods $1, 3, 5, \ldots$, and even-period schedules in periods $2, 4, 6, \ldots$. Hence, $[0, \tau_o], [\tau_o + \tau_e, 2\tau_o + \tau_e], [2(\tau_o + \tau_e), 2(\tau_o + \tau_e) +$

$\tau_o], \ldots$ are the odd-period intervals, while $[\tau_o, \tau_o + \tau_e], [2\tau_o + \tau_e, 2(\tau_o + \tau_e)], [3\tau_o + 2\tau_e, 3(\tau_o + \tau_e)], \ldots$ are the even-period intervals. For example, a job that is partly processed in period 1 cannot be processed further in period 2 or any even periods; however, it can be processed further in period 3 or any other odd periods. Job preemption is allowed in this model. This model has some similarities with the classical two-parallel-machine scheduling model if we view the odd periods and even periods as two different machines. However, its solvability is different from the classical two-parallel-machine scheduling model. For example, we show in Section 2.2 below that problem $1 \mid alt \mid \sum C_j$ is binary *NP*-hard, whereas the classical problem $P2||\sum C_j$ is solvable in polynomial time [10]. In the following subsections, we analyze problems $1 \mid alt \mid \sum w_j C_j$, $1 \mid alt \mid \sum C_j$, $1 \mid alt \mid L_{\max}$, and $1 \mid alt \mid \sum U_j$.

2.1 Total weighted completion time

We first illustrate problem $1 \mid alt \mid \sum w_j C_j$ with the following example: $\tau_o = \tau_e = 10$, $n = 5$, $(w_1, w_2, w_3, w_4, w_5) = (2, 2, 2, 1, 1)$, and $(p_1, p_2, p_3, p_4, p_5) = (4, 7, 11, 6, 9)$. A feasible schedule is shown in Figure 1, where jobs 1, 4, and 5 are processed in the odd periods, and jobs 2 and 3 are processed in the even periods. The total weighted completion time is $w_1C_1 + w_2C_2 + w_3C_3 + w_4C_4 + w_5C_5 = (2)(4) + (2)(17) + (2)(38) + (1)(10) + (1)(29) = 157$.

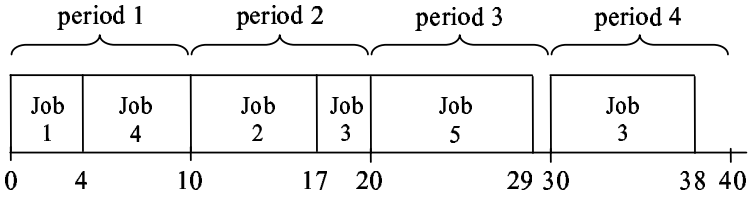


Figure 1: Example of Problem $1 \mid alt \mid \sum w_j C_j$.

In order to analyze the solvability of the problem with a total weighted completion time objective, we need the following preliminary result.

Lemma 1 *Let $x_1, \ldots, x_m, z$ be $m+1$ nonnegative integers which satisfy (i) $x_1 + \cdots + x_m = mz$, (ii) $x_1 + \cdots + x_i \leq iz$ for $i = 1, \ldots, m$, and (iii) $1x_1 + 2x_2 + \cdots + mx_m \leq \frac{1}{2}m(m+1)z$. Then, $x_1 = x_2 = \cdots = x_m = z$.*

Proof. We first prove that, under conditions (i) and (ii), the quantity "$1x_1 + 2x_2 + \cdots + mx_m$" is minimized if and only if $x_1 = x_2 = \cdots = x_m = z$. Suppose, to the contrary, that

$1x_1 + 2x_2 + \cdots + mx_m$ is minimized, but there exists $i \in \{1, \ldots, m\}$ such that $x_i \neq z$. Let $g = \min\{i \mid x_i \neq z\}$. Clearly, $x_g < z$, otherwise (ii) is violated. Condition (i) implies that there exists $i \in \{g+1, \ldots, m\}$ such that $x_i > z$. Let $h = \min\{i \mid x_i > z\}$. Increasing x_g by 1 and decreasing x_h by 1 simultaneously decreases the value of $1x_1 + 2x_2 + \cdots + mx_m$ without violating conditions (i) and (ii), a contradiction. Hence, $1x_1 + 2x_2 + \cdots + mx_m$ is not minimized unless $x_1 = x_2 = \cdots = x_m = z$. This implies that the minimum possible value of $1x_1 + 2x_2 + \cdots + mx_m$ is $1z + 2z + \cdots + mz = \frac{1}{2}m(m+1)z$, which is attainable only when $x_1 = x_2 = \cdots = x_m = z$. Therefore, from (iii), we have $x_1 = x_2 = \cdots = x_m = z$. □

We now provide the main result of this subsection.

Theorem 1 *The recognition version of problem* $1 \mid alt \mid \sum w_j C_j$ *is unary* NP*-complete.*

Proof. By reduction from the following problem, which is unary *NP*-complete [5]:

3-Partition: Given integers $a_1, \ldots, a_{3m}$, where $\sum_{j=1}^{3m} a_j = mb$ and $b/4 < a_j < b/2$ for $j = 1, \ldots, 3m$, does there exist a partition $\{S_1, \ldots, S_m\}$ of $\{1, \ldots, 3m\}$ such that $|S_i| = 3$ and $\sum_{j \in S_i} a_j = b$, for $i = 1, \ldots, m$?

Given an instance of 3-Partition, we construct an instance of problem $1 \mid alt \mid \sum w_j C_j$, as follows:

$n = 3m + 1$;

$p_j = w_j = M + a_j$, where $M = \frac{1}{2}m(m+1)b + 1$, for $j = 1, \ldots, 3m$;

$p_{3m+1} = (m+1)\tau$, where $\tau = 3M + b$;

$w_{3m+1} = \frac{1}{2}m(2m+1)\tau^2 + \frac{1}{2}\sum_{j=1}^{3m}(M + a_j)^2 + 1$;

$\tau_o = \tau_e = \tau$;

$C = \frac{1}{2}m(2m+1)\tau^2 + \frac{1}{2}\sum_{j=1}^{3m}(M + a_j)^2 + w_{3m+1}(2m+1)\tau$.

Clearly, this construction requires polynomial time. We show that there exists a solution to this instance of problem $1 \mid alt \mid \sum w_j C_j$ with $\sum w_j C_j \leq C$ if and only if there exists a solution to the given instance of 3-Partition.

($\Rightarrow$) Let $S_i = \{\pi_{3i-2}, \pi_{3i-1}, \pi_{3i}\}$, $i = 1, \ldots, m$, denote a solution to the given instance of 3-Partition, where $(\pi_1, \ldots, \pi_{3m})$ is a permutation of $(1, \ldots, 3m)$. Consider the schedule

$$\sigma = (3m+1, \pi_1, \pi_2, \pi_3, 3m+1, \pi_4, \pi_5, \pi_6, 3m+1, \ldots, 3m+1, \pi_{3m-2}, \pi_{3m-1}, \pi_{3m}, 3m+1),$$

where each portion of job $3m+1$ has length τ. In this schedule, job $3m+1$ is processed in the odd periods, whereas the other jobs are processed in the even periods. The completion

times of jobs π_{3i-2}, π_{3i-1}, and π_{3i} are $i\tau + \sum_{j=1}^{3i-2}(M + a_{\pi_j})$, $i\tau + \sum_{j=1}^{3i-1}(M + a_{\pi_j})$, and $i\tau + \sum_{j=1}^{3i}(M + a_{\pi_j})$, respectively, for $i = 1, \ldots, m$. Thus, the total weighted completion time of jobs $1, \ldots, 3m$ is

$$\sum_{i=1}^{m}\left\{(M+a_{\pi_{3i-2}})\left[i\tau+\sum_{j=1}^{3i-2}(M+a_{\pi_j})\right]+(M+a_{\pi_{3i-1}})\left[i\tau+\sum_{j=1}^{3i-1}(M+a_{\pi_j})\right]+(M+a_{\pi_{3i}})\left[i\tau+\sum_{j=1}^{3i}(M+a_{\pi_j})\right]\right\}$$
$$= \sum_{i=1}^{m} i\tau(3M + a_{\pi_{3i-2}} + a_{\pi_{3i-1}} + a_{\pi_{3i}}) + \sum_{j=1}^{3m}\sum_{r=1}^{j}(M + a_j)(M + a_r)$$
$$= \tau^2\sum_{i=1}^{m} i + \sum_{j=1}^{3m}\sum_{r=1}^{j}(M + a_j)(M + a_r)$$
$$= \frac{1}{2}m(m+1)\tau^2 + \frac{1}{2}\left[\sum_{j=1}^{3m}(M + a_j)\right]^2 + \frac{1}{2}\sum_{j=1}^{3m}(M + a_j)^2$$
$$= \frac{1}{2}m(2m+1)\tau^2 + \frac{1}{2}\sum_{j=1}^{3m}(M + a_j)^2.$$

The weighted completion time of job $3m + 1$ is $w_{3m+1}(2m + 1)\tau$. Hence, $\sum w_jC_j = C$.

($\Leftarrow$) Suppose that there exists a feasible schedule σ for the above instance of problem $1 \mid alt \mid \sum w_jC_j$ with $\sum w_jC_j \leq C$. Since all job processing times are integer, there must exist a feasible schedule σ' with $\sum w_jC_j \leq C$ where all job completion times are integer. Since $w_{3m+1}[(2m + 1)\tau + 1] > C$, schedule σ' must process job $3m + 1$ consecutively in the odd-period schedule; that is, job $3m+1$ is processed in periods $[0, \tau], [2\tau, 3\tau], \ldots, [2m\tau, (2m+1)\tau]$. This leaves periods $[\tau, 2\tau], [3\tau, 4\tau], \ldots, [(2m - 1)\tau, 2m\tau]$ for the processing of jobs $1, \ldots, 3m$.

We observe that, for the $\sum w_jC_j$ objective, it is not possible to reduce cost by allowing one job to preempt another. Consequently, there must also exist a feasible schedule σ'' such that (i) $\sum w_jC_j \leq C$, (ii) jobs $1, \ldots, 3m$ are processed in periods $[\tau, 2\tau], [3\tau, 4\tau], \ldots, [(2m - 1)\tau, 2m\tau]$, and (iii) jobs $1, \ldots, 3m$ do not preempt each other (i.e., they only preempt job $3m + 1$). Let $F(\sigma'')$ denote the total weighted completion time of the jobs in schedule σ''.

Consider jobs $1, \ldots, 3m$ in schedule σ''. For $i = 1, \ldots, m$, let $S_i = \{x_{i,1}, \ldots, x_{i,|S_i|}\}$ denote the subset of jobs that finish processing in the time interval $[(2i - 1)\tau, 2i\tau]$, where $x_{i,l}$ is processed before $x_{i,l+1}$ for $l = 1, \ldots, |S_i| - 1$. Note that

$$|S_1| + |S_2| + \cdots + |S_m| = 3m. \tag{1}$$

For $i = 1, \ldots, m$, because $M > ib$, we have $(3i + 1)M > i\tau$. Thus, since each job has a processing time greater than M, it is impossible to finish processing more than $3i$ jobs in

total in the time intervals $[\tau, 2\tau], [3\tau, 4\tau], \ldots, [(2i-1)\tau, 2i\tau]$. Hence,

$$|S_1| + |S_2| + \cdots + |S_i| \leq 3i \text{ for } i = 1, \ldots, m. \tag{2}$$

For $i = 1, \ldots, m$ and $l = 1, \ldots, |S_i|$, the completion time of job $x_{i,l}$ is $i\tau + \sum_{r=1}^{i-1} \sum_{s=1}^{|S_r|}(M + a_{x_{r,s}}) + \sum_{s=1}^{l}(M + a_{x_{i,s}})$. Thus,

$$\begin{aligned}
F(\sigma'') &= \sum_{i=1}^{m} \sum_{l=1}^{|S_i|}(M + a_{x_{i,l}}) \left[i\tau + \sum_{r=1}^{i-1} \sum_{s=1}^{|S_r|}(M + a_{x_{r,s}}) + \sum_{s=1}^{l}(M + a_{x_{i,s}}) \right] + w_{3m+1}(2m+1)\tau \\
&= \sum_{i=1}^{m} i\tau \sum_{l=1}^{|S_i|}(M + a_{x_{i,l}}) + \sum_{j=1}^{3m} \sum_{r=1}^{j}(M + a_j)(M + a_r) + w_{3m+1}(2m+1)\tau \\
&= \tau M \sum_{i=1}^{m} i|S_i| + \sum_{i=1}^{m} i\tau \sum_{l=1}^{|S_i|} a_{x_{i,l}} + \frac{1}{2}\Big[\sum_{j=1}^{3m}(M + a_j) \Big]^2 + \frac{1}{2} \sum_{j=1}^{3m}(M + a_j)^2 + w_{3m+1}(2m+1)\tau \\
&= \tau M \sum_{i=1}^{m} i|S_i| + \sum_{i=1}^{m} i\tau \sum_{l=1}^{|S_i|} a_{x_{i,l}} + \frac{1}{2} m^2\tau^2 + \frac{1}{2} \sum_{j=1}^{3m}(M + a_j)^2 + w_{3m+1}(2m+1)\tau.
\end{aligned}$$

Since $F(\sigma'') \leq C$, we have

$$\tau M \sum_{i=1}^{m} i|S_i| + \sum_{i=1}^{m} i\tau \sum_{l=1}^{|S_i|} a_{x_{i,l}} + \frac{1}{2} m^2\tau^2 \leq \frac{1}{2} m(2m+1)\tau^2.$$

After simplification, we have

$$M \sum_{i=1}^{m} i|S_i| + \sum_{i=1}^{m} i \sum_{l=1}^{|S_i|} a_{x_{i,l}} \leq \frac{1}{2} m(m+1)\tau. \tag{3}$$

Hence,

$$M \sum_{i=1}^{m} i|S_i| \leq \frac{3}{2} m(m+1)M + \frac{1}{2} m(m+1)b.$$

Since M, $\sum_{i=1}^{m} i|S_i|$, and $\frac{3}{2}m(m+1)$ are integers and $M > \frac{1}{2}m(m+1)b$, we have

$$\sum_{i=1}^{m} i|S_i| \leq \frac{3}{2} m(m+1). \tag{4}$$

From (1), (2), (4), and Lemma 1, we conclude that $|S_1| = \cdots = |S_m| = 3$. Thus, inequality (3) can be rewritten as

$$M \sum_{i=1}^{m} 3i + \sum_{i=1}^{m} i \sum_{l=1}^{3} a_{x_{i,l}} \leq \frac{1}{2} m(m+1)\tau,$$

which implies that

$$\sum_{i=1}^{m} i \sum_{l=1}^{3} a_{x_{i,l}} \leq \frac{1}{2} m(m+1)b. \tag{5}$$

Observe that the total processing time of the jobs in $S_1 \cup \cdots \cup S_i$ is at most $i\tau$, and that the total processing time of the jobs in $S_1 \cup \cdots \cup S_m$ is equal to $m\tau$. Hence,

$$\sum_{r=1}^{i} \sum_{l=1}^{3} a_{x_{r,l}} \leq ib \text{ for } i = 1, \ldots, m \tag{6}$$

and

$$\sum_{i=1}^{m} \sum_{l=1}^{3} a_{x_{i,l}} = mb. \tag{7}$$

Then, from (5), (6), (7), and Lemma 1, we have

$$\sum_{l=1}^{3} a_{x_{i,l}} = b$$

for $i = 1, \ldots, m$. Therefore, $\{S_1, \ldots, S_m\}$ forms a 3-partition of $\{1, \ldots, 3m\}$. □

2.2 Total completion time

We first present a property of the problem with total completion time objective.

Lemma 2 *In problem* $1 \mid alt \mid \sum C_j$*, there exists an optimal schedule in which the jobs within each of the odd-period and even-period schedules are processed in SPT order with no inserted idle time in that schedule.*

Proof. Clearly, in any optimal schedule, there is no inserted idle time in the odd-period schedule and in the even-period schedule. Suppose, in contradiction of the SPT order, that there exists an optimal schedule σ in which there are two jobs i and j, such that (i) these two jobs are in the same, either odd-period or even-period, schedule; (ii) $p_i < p_j$; and (iii) job j immediately precedes job i. Then, consider an alternative schedule σ', where jobs i and j are interchanged but they both remain within the same schedule. Observe that $C_j(\sigma') = C_i(\sigma)$, where both completion times include the same number of interruptions of processing, during the processing of the other schedule. Furthermore, since $p_i < p_j$, we have $C_i(\sigma') < C_j(\sigma)$, where the number of interruptions of processing before $C_i(\sigma')$ is no larger than that before $C_j(\sigma)$. Hence, $\sum C_j(\sigma') < \sum C_j(\sigma)$, which contradicts the optimality of schedule σ. □

Based on Lemma 2, we propose the following dynamic programming algorithm for problem $1 \mid alt \mid \sum C_j$. In this dynamic program, we assign some jobs to the odd periods and

some jobs to the even periods. A job that cannot complete within an odd (respectively, even) period is continued in the next odd (respectively, even) period.

Algorithm AltC

Preprocessing:

Index the jobs in SPT order, i.e., $p_1 \leq \cdots \leq p_n$.

Optimal value function:

$f_j(t_o)$ = minimum total completion time of a schedule of jobs $1, \ldots, j$ that schedules total time t_o in the odd-period schedule, for $j = 0, 1, \ldots, n$ and $t_o = 0, 1, \ldots, \sum_{i=1}^{j} p_i$.

Boundary conditions:

$f_0(0) = 0$.

$f_j(t_o) = +\infty$ if $t_o < 0$ or $t_o > \sum_{i=1}^{j} p_i$, for $j = 0, 1, \ldots, n$.

Recurrence relation:

For $j = 1, \ldots, n$ and $t_o = 0, 1, \ldots, \sum_{i=1}^{j} p_i$,

$$f_j(t_o) = \min\left\{f_{j-1}(t_o - p_j) + \left\lceil \frac{t_o - \tau_o}{\tau_o} \right\rceil \tau_e + t_o,\ f_{j-1}(t_o) + \left(\left\lceil \frac{\sum_{i=1}^{j} p_i - t_o - \tau_e}{\tau_e} \right\rceil + 1 \right) \tau_o + \sum_{i=1}^{j} p_i - t_o \right\}.$$

Optimal solution value:

$\min_{0 \leq t_o \leq P}\{f_n(t_o)\}$.

In the recurrence relation, the first term of the minimization is the cost of including job j in the odd-period schedule, where "$\lceil (t_o - \tau_o)/\tau_o \rceil \tau_e + t_o$" is the completion time of job j, which includes the amount of time $\lceil (t_o - \tau_o)/\tau_o \rceil \tau_e$ in the even periods and the amount of time t_o in the odd periods. The second term is the cost of including job j in the even-period schedule, where "$(\lceil (\sum_{i=1}^{j} p_i - t_o - \tau_e)/\tau_e \rceil + 1)\tau_o + \sum_{i=1}^{j} p_i - t_o$" is the completion time of job j, which includes the amount of time $(\lceil (\sum_{i=1}^{j} p_i - t_o - \tau_e)/\tau_e \rceil + 1)\tau_o$ in the odd periods and the amount of time $\sum_{i=1}^{j} p_i - t_o$ in the even periods.

We now present the main result of this subsection.

Theorem 2 *Algorithm AltC finds an optimal schedule for problem* $1 \mid alt \mid \sum C_j$ *in* $O(nP)$ *time.*

Proof. The sequencing of the jobs in SPT order in both the odd-period and even-period schedules is justified by Lemma 2. The recurrence relation compares the cost of scheduling

the next shortest processing time job within either the odd-period or the even-period schedule, and therefore compares the cost of all possible schedules. The optimal value function is computed for $j = 1, \ldots, n$ and $t_o = 0, 1, \ldots, P$. If we precompute the partial sums $\sum_{i=1}^{j} p_i$, for $j = 1, \ldots, n$, then each application of the recurrence relation requires only constant time. Therefore, the overall computational requirement of Algorithm AltC is $O(nP)$ time. □

The running time of Algorithm AltC is pseudopolynomial. The following result shows that the existence of a polynomial time algorithm for problem $1 \mid alt \mid \sum C_j$ is unlikely.

Theorem 3 *The recognition version of problem* $1|alt|\sum C_j$ *is binary* NP*-complete.*

Proof. By reduction from the following problem, which is binary *NP*-complete [5]:

Even-Odd Partition: Given integers a_{2i-1} and a_{2i} for $i = 1, \ldots, m$, where $a_1 < \cdots < a_{2m}$ and $\sum_{i=1}^{2m} a_i = 2b$, does there exist a partition $\{S_1, S_2\}$ of $\{1, \ldots, 2m\}$ such that a_{2i-1} and a_{2i} are in different sides of the partition for $i = 1, \ldots, m$ and that $\sum_{j \in S_1} a_j = \sum_{j \in S_2} a_j = b$?

For a given instance of the Even-Odd Partition problem, let $\Delta = \frac{1}{2} \sum_{j=1}^{m} (a_{2j} - a_{2j-1})$. Without loss of generality, we may assume that the given instance satisfies the constraints that $a_2 > a_1 > \Delta$ and $a_{2i} > a_{2i-1} > \sum_{j=1}^{2i-2} a_j$ for each $2 \leq i \leq m$. If the instance does not satisfy the constraints, then we construct another instance $\{a'_1, a'_2, \ldots, a'_{2m}\}$ that satisfies the constraints. The construction is performed as follows (see [4] for a similar construction): $a'_1 = a_1 + \Delta$, $a'_2 = a_2 + \Delta$, and for each $i = 2, \ldots, m$, $a'_{2i-1} = a_{2i-1} + \sum_{j=1}^{2i-2} a'_j$ and $a'_{2i} = a_{2i} + \sum_{j=1}^{2i-2} a'_j$. Clearly, the construction can be performed in polynomial time, and the two instances have the same answer. In the following, we assume that the given instance of Even-Odd Partition satisfies the constraints.

Given an instance of Even-Odd Partition which satisfies the above constraints, we consider the following instance of problem $1|alt|\sum C_j$, where $M = \sum_{i=1}^{m} (m - i + 1)(a_{2i-1} + a_{2i})$:

$n = 2m$;

$p_j = M + a_j$, for $j = 1, \ldots, 2m$;

$\tau_o = \tau_e = \tau$, where $\tau = mM + b$;

$C = m\tau + m(m+1)M + M$.

Clearly, this construction requires polynomial time. We show that there exists a solution to this instance of problem $1|alt|\sum C_j$ with $\sum C_j \leq C$ if and only if there exists a solution to the given instance of Even-Odd Partition.

($\Rightarrow$) Let $S_1 = \{s'_1, \ldots, s'_m\}$ and $S_2 = \{s''_1, \ldots, s''_m\}$ denote a solution to the given instance of Even-Odd Partition, where $\{s'_i, s''_i\} = \{2i-1, 2i\}$ for $i = 1, \ldots, m$. Consider the schedule

$$\sigma = (s'_1, \ldots, s'_m, s''_1, \ldots, s''_m)$$

with no idle time between the jobs. Note that $\sum_{i=1}^{m} p_{s'_i} = mM + \sum_{i=1}^{m} a_{s'_i} = mM + \sum_{j \in S_1} a_j = \tau$ and $\sum_{i=1}^{m} p_{s''_i} = mM + \sum_{i=1}^{m} a_{s''_i} = mM + \sum_{j \in S_2} a_j = \tau$. Thus, jobs $s'_1, \ldots, s'_m$ are processed within the odd period $[0, \tau_o]$, and jobs $s''_1, \ldots, s''_m$ are processed within the even period $[\tau_o, \tau_o + \tau_e]$. Furthermore,

$$\begin{aligned}
\sum_{j=1}^{n} C_j &= \sum_{j \in S_1} C_j + \sum_{j \in S_2} C_j \\
&= \left[\sum_{i=1}^{m} (m-i+1) p_{s'_i} \right] + \left[m\tau + \sum_{i=1}^{m} (m-i+1) p_{s''_i} \right] \\
&= m\tau + \sum_{i=1}^{m} (m-i+1)(M + a_{s'_i}) + \sum_{i=1}^{m} (m-i+1)(M + a_{s''_i}) \\
&= m\tau + m(m+1)M + \sum_{i=1}^{m} (m-i+1)(a_{s'_i} + a_{s''_i}) = C.
\end{aligned}$$

($\Leftarrow$) Suppose that there exists a feasible schedule for the above instance of problem $1|alt|\sum C_j$ with $\sum C_j \leq C$. Since $\tau < (m+1)M$ and $p_j > M$ for $j = 1, \ldots, 2m$, at most m jobs can start and end within a period. If m jobs are processed within the period $[0, \tau]$ and m jobs are processed within the period $[\tau, 2\tau]$, then the total completion time of jobs is greater than $\sum_{i=1}^{m} iM + \sum_{i=1}^{m} (\tau + iM) = m\tau + m(m+1)M$, regardless of which jobs are processed in each period. Further, if any job completes after time 2τ, this increases the total completion time by at least $\tau > M$. Thus, if there is a job completed after time 2τ, the total completion time of jobs is greater than $m\tau + m(m+1)M + M = C$. Hence, exactly m jobs are processed within the time period $[0, \tau]$, and exactly m jobs are processed within the time period $[\tau, 2\tau]$. Let S_1 and S_2 denote the set of jobs scheduled in the first and second period, respectively. Therefore, $\sum_{j \in S_1} p_j = \sum_{j \in S_2} p_j = mM + b$, which implies that $\sum_{j \in S_1} a_j = \sum_{j \in S_2} a_j = b$.

Finally, we show that, for each $i = 1, \ldots, m$, the jobs $2i-1$ and $2i$ cannot both be processed within the same period. Assume to the contrary that k denotes the largest index such that the jobs $2k-1$ and $2k$ are both scheduled in the same period. Thus, for $i = k+1, \ldots, m$, jobs $2i-1$ and $2i$ are scheduled in different periods. In the period where both jobs are processed, there are $k-2$ jobs processed before jobs $2k-1$ and $2k$, and each of

these $k-2$ jobs has a processing time greater than M, where $k \geq 2$. Between jobs $2i-1$ and $2i$, let s_i denote the job that is scheduled in the same period as jobs $2k-1$ and $2k$, for $i = k+1, \ldots, m$. Then, the total processing time of jobs scheduled in this period is at least

$$\begin{aligned}
(k-2)M + p_{2k-1} + p_{2k} + \sum_{i=k+1}^{m} p_{s_i} &= mM + a_{2k-1} + a_{2k} + \sum_{i=k+1}^{m} a_{s_i} \\
&\geq mM + a_{2k} + \sum_{i=k}^{m} a_{2i-1} \\
&> mM + \sum_{j=1}^{2k-2} a_j + \sum_{i=k}^{m} a_{2i-1} \\
&\geq mM + a_2 + \sum_{i=1}^{m} a_{2i-1} \\
&= mM + a_2 + \frac{1}{2}\sum_{i=1}^{m}(a_{2i-1} + a_{2i}) - \Delta \\
&> mM + \frac{1}{2}\sum_{i=1}^{m}(a_{2i-1} + a_{2i}) \\
&= mM + b,
\end{aligned}$$

where the second inequality follows from the constraint "$a_{2i} > \sum_{j=1}^{2i-2} a_j$," and the fourth inequality follows from the constraint "$a_2 > \Delta$." This implies that the m jobs cannot complete their processing within this period, a contradiction. Therefore, $\{S_1, S_2\}$ constitutes a solution to the Even-Odd Partition problem. □

2.3 Maximum lateness

We first present a property of the problem with maximum lateness objective.

Lemma 3 *In problem* $1 \mid alt \mid L_{\max}$*, there exists an optimal schedule in which the jobs within each of the odd-period and even-period schedules are processed in EDD order with no inserted idle time in that schedule.*

Proof. Clearly, in any optimal schedule, there is no inserted idle time in the odd-period schedule and in the even-period schedule. Suppose that there exists an optimal schedule σ in which there are two jobs i and j, such that (i) these two jobs are in the same, either odd-period or even-period, schedule; (ii) $d_i < d_j$; and (iii) job j immediately precedes job i. Then, consider an alternative schedule σ', where jobs i and j are interchanged but both remain within the same schedule. Observe that $C_j(\sigma') = C_i(\sigma)$, where both completion

times include the same number of interruptions of processing, during the processing of the other schedule. Furthermore, since $d_i < d_j$, we have $L_j(\sigma') < L_i(\sigma)$. Clearly, $L_i(\sigma') < L_i(\sigma)$. Thus, $L_{\max}(\sigma') \leq L_{\max}(\sigma)$. Hence, σ' is also optimal. Repeating this process, we obtain an optimal schedule in which the jobs within each of the odd-period and even-period schedules are arranged in EDD order. □

Based on Lemma 3, we propose the following dynamic programming algorithm for problem $1 \mid alt \mid L_{\max}$.

Algorithm AltL

Preprocessing:

Index the jobs in EDD order, i.e., $d_1 \leq \cdots \leq d_n$.

Optimal value function:

$f_j(t)$ = minimal maximum lateness of a schedule of jobs $1, \ldots, j$ that schedules total time t in the odd-period schedule, for $j = 0, 1, \ldots, n$ and $t = 0, 1, \ldots, \sum_{i=1}^{j} p_i$.

Boundary conditions:

$f_0(0) = -\infty$.

$f_j(t) = +\infty$ if $t < 0$ or $t > \sum_{i=1}^{j} p_i$, for $j = 0, 1, \ldots, n$.

Recurrence relation:

For $j = 1, \ldots, n$ and $t = 0, 1, \ldots, \sum_{i=1}^{j} p_i$,

$$f_j(t) = \min\Big\{ \max\{f_{j-1}(t - p_j), \lceil (t - \tau_o)/\tau_o \rceil \tau_e + t - d_j\}, \\ \max\{f_{j-1}(t), (\lceil (\textstyle\sum_{i=1}^{j} p_i - t - \tau_e)/\tau_e \rceil + 1)\tau_o + \sum_{i=1}^{j} p_i - t - d_j\}\Big\}.$$

Optimal solution value:

$\min_{0 \leq t \leq P}\{f_n(t)\}$.

In the recurrence relation, the quantity "$\max\{f_{j-1}(t-p_j), \lceil (t-\tau_o)/\tau_o \rceil \tau_e + t - d_j\}$" is the cost of including job j in the odd-period schedule. In this expression, "$\lceil (t - \tau_o)/\tau_o \rceil \tau_e + t$" is the completion time of job j, which includes the amount of time "$\lceil (t - \tau_o)/\tau_o \rceil \tau_e$" in the even periods and the amount of time t in the odd periods. Thus, "$\lceil (t - \tau_o)/\tau_o \rceil \tau_e + t - d_j$" is the lateness of job j. Similarly, the quantity "$\max\{f_{j-1}(t), (\lceil (\sum_{i=1}^{j} p_i - t - \tau_e)/\tau_e \rceil + 1)\tau_o + \sum_{i=1}^{j} p_i - t - d_j\}$" is the lateness cost of including job j in the even-period schedule.

We now present the main result of this subsection.

Theorem 4 *Algorithm AltL finds an optimal schedule for problem* $1 \mid alt \mid L_{\max}$ *in* $O(nP)$ *time.*

Proof. Similar to the proof of Theorem 2. □

The running time of Algorithm AltL is pseudo-polynomial. The following result shows that the existence of a polynomial time algorithm for problem $1 \mid alt \mid L_{\max}$ is unlikely.

Theorem 5 *The recognition version of problem* $1 \mid alt \mid L_{\max}$ *is binary* NP*-complete.*

Proof. By reduction from the following problem, which is binary *NP*-complete [5].
Partition: Given integers $a_1, \ldots, a_m$ where $\sum_{i=1}^{m} a_i = 2b$, does there exist a partition $\{S_1, S_2\}$ of $\{1, \ldots, m\}$ where $\sum_{j \in S_1} a_j = \sum_{i \in S_2} a_j = b$?

Given an instance of Partition, we consider an instance of problem $1 \mid alt \mid L_{\max}$ defined as follows:
$n = m$;
$p_j = a_j$, for $j = 1, \ldots, n$;
$d_j = 2b$, for $j = 1, \ldots, n$;
$\tau_o = \tau_e = b$;
$L = 0$.

There exists a solution to this instance of problem $1 \mid alt \mid L_{\max}$ with $L_{\max} \leq L$ if and only if all n jobs are processed within the time interval $[0, \tau_o + \tau_e]$, i.e., if and only if each job is processed either in the interval $[0, \tau_o]$ or in the interval $[\tau_o, \tau_o + \tau_e]$. Hence, there exists a solution to this instance of problem $1 \mid alt \mid L_{\max}$ with $L_{\max} \leq L$ if and only if there exists a solution to the given instance of Partition. □

2.4 Number of late jobs

To solve problem $1 \mid alt \mid \sum U_j$, we place the late jobs at the end of the schedule. The on-time jobs need to be completed by their due dates. Some of the on-time jobs are assigned to the odd periods, and some of them are assigned to the even periods. We propose the following dynamic programming algorithm:

Algorithm AltU

Preprocessing:

Index the jobs in EDD order, i.e., $d_1 \leq \cdots \leq d_n$.

Optimal value function:

$f_j(k, t_o)$ = minimum total processing time of on-time jobs in the even periods of a schedule from jobs $1, \ldots, j$, such that exactly k of those jobs are late and the total processing time of on-time jobs in the odd periods is t_o, where $f_j(k, t_o) = +\infty$ if no such schedule exists, for $j = 0, 1, \ldots, n$; $k = 0, 1, \ldots, j$; $t_o = 0, 1, \ldots, \sum_{i=1}^{j} p_i$.

Boundary conditions:

$f_0(0, 0) = 0$.

$f_0(0, t_o) = +\infty$ if $t_o \neq 0$.

$f_j(k, t_o) = +\infty$ if $k > j$ or $t_o < 0$ or $t_o + (\lceil \frac{t_o}{\tau_o} \rceil - 1)\tau_e > d_j$.

Recurrence relation:

For $j = 1, \ldots, n$, $k = 0, 1, \ldots, j$, and $t_o = 0, 1, \ldots, \sum_{i=1}^{j} p_i$ such that $t_o + (\lceil \frac{t_o}{\tau_o} \rceil - 1)\tau_e \leq d_j$,

$$f_j(k, t_o) = \min\{f_{j-1}(k-1, t_o), f_{j-1}(k, t_o - p_j), e_j(k, t_o)\},$$

where

$$e_j(k, t_o) = \begin{cases} f_{j-1}(k, t_o) + p_j, & \text{if } f_{j-1}(k, t_o) + p_j + \lceil (f_{j-1}(k, t_o) + p_j)/\tau_e \rceil \tau_o \leq d_j; \\ +\infty, & \text{otherwise.} \end{cases}$$

Optimal solution value:

$\min\{k \mid t_o + f_n(k, t_o) \leq P \text{ for some } t_o = 0, 1, \ldots\}$.

In the recurrence relation, the condition "$t_o + (\lceil \frac{t_o}{\tau_o} \rceil - 1)\tau_e \leq d_j$" ensures that the completion of the last on-time job in the odd-period schedule does not exceed d_j. The first term of the minimization is for the case where job j is a late job. The second (respectively, third) term is for the case where job j is an on-time job and scheduled in an odd (respectively, even) period. The last case is only available if job j completes on time when it is scheduled in an even period, i.e., if $f_{j-1}(k, t_o) + p_j + \lceil (f_{j-1}(k, t_o) + p_j)/\tau_e \rceil \tau_o \leq d_j$.

We now present the main result of this subsection.

Theorem 6 *Algorithm AltU finds an optimal schedule for problem* $1 \mid alt \mid \sum U_j$ *in* $O(n^2 P)$ *time.*

Proof. The sequencing of the on-time jobs in EDD order in both the odd-period and even-period schedules is justified by Lemma 3. The recurrence relation compares the cost of scheduling the next job to be a late job, or to be an on-time job scheduled in an odd period, or to be an on-time job scheduled in an even period. Therefore, it compares the cost of all possible schedules. The optimal value function is computed for $j = 1, \ldots, n$, $k = 0, 1, \ldots, j$, and $t_o = 0, 1, \ldots, \sum_{i=1}^{j} p_i$, and each computation requires constant time. Therefore, the overall computational requirement of Algorithm AltU is $O(n^2 P)$ time. □

The running time of Algorithm AltU is pseudo-polynomial. The following result shows that the existence of a polynomial time algorithm for problem $1 \mid alt \mid \sum U_j$ is unlikely.

Corollary 1 *The recognition version of problem* $1 \mid alt \mid \sum U_j$ *is binary* NP*-complete.*

Proof. Follows from Theorem 5. □

3 Shared Processing with Routine Activities

In our second system design, we are given a set of n primary jobs $N = \{1, \ldots, n\}$ and a set of $\bar{n}$ routine jobs $\bar{N} = \{n+1, \ldots, n+\bar{n}\}$. If a primary job is in progress when a time period is reached at which a routine job is required, then processing is shared between the two jobs. The share of processing given to the continuation of the primary job is denoted by e, and the remaining share $1 - e$ is used for the routine job, where e is a rational number such that $0 \leq e \leq 1$. We let r_j denote the time at which routine job j becomes due for processing.

The primary jobs are all available for processing at time $t = 0$, but the routine jobs have different release times. For the applications mentioned in Section 1, their release times are known in advance. If the primary job completes before the routine job, then we immediately start the next primary job so that it will share the processor with the routine job. On the other hand, if the routine job completes before the primary job, the primary job will have access to a full processor unless there is another routine job waiting for processing. No primary job can preempt another primary job, and no routine job can preempt another routine job. Without loss of generality, we assume that the routine jobs are serviced according to a first-come-first-serve rule. That is, if two routine jobs arrive at almost the same time,

then the job that arrives first is processed first. When it completes, the second routine job is processed. The primary jobs follow some sequence which we discuss below.

In many practical situations, the relative priority given to routine activities is determined by established rule. This is because those activities are essential to the maintenance of the processing system, and also because the value of $e > 0$ represents the urgency of the primary jobs. For example, in systems where the on-time performance of routine jobs is critical, a relatively low value of e is appropriate. Hence, we model the value of e as a constant, independent of the set of primary jobs that define an instance. This assumption is generally used when modeling interruption by routine work as a vacation in a queueing system [3, 22]. The requirement of processor sharing ensures the processing of the routine jobs with reasonable promptness. We observe that the completion times of the routine jobs are independent of the sequencing decision of the primary jobs. Hence, we do not include the completion time cost of the routine jobs in our objective.

3.1 Total weighted completion time

We consider the solvability of problem $1 \mid share(e) \mid \sum w_j C_j$, for any given $e \in [0, 1]$. We begin with comments about two special cases of the problem. First, when $e = 0$, the problem becomes a scheduling problem with machine availability constraints and a total weighted completion time objective. Wang et al. [23] show that this problem is unary *NP*-hard. However, their proof does not apply here when $e > 0$. Second, when $e = 1$, the problem is easily solved. All the primary jobs are scheduled without interruption according to the SWPT rule, and then following their completion, the routine jobs are scheduled.

It is somewhat intuitive that the SWPT rule may sequence the primary jobs optimally for $1 \mid share(e) \mid \sum w_j C_j$. However, the following example shows that this is not the case.

Example 1: Consider two primary jobs (jobs 1 and 2) and one routine job (job 3). The sharing proportion is $e = 0.5$. Job 1 has $w_1 = 1$ and $p_1 = 1$. Job 2 has $w_2 = 2.1$ and $p_2 = 2$. Job 3's release time is 1 and its processing requirement is 1. The SWPT rule schedules job 2 before job 1, with $\sum w_j C_j = (2.1)(3) + (1)(4) = 10.3$ (see Figure 2(a)). An optimal solution schedules job 1 before job 2, with $\sum w_j C_j = (1)(1) + (2.1)(4) = 9.4$ (see Figure 2(b)). Hence, the SWPT schedule is not optimal. □

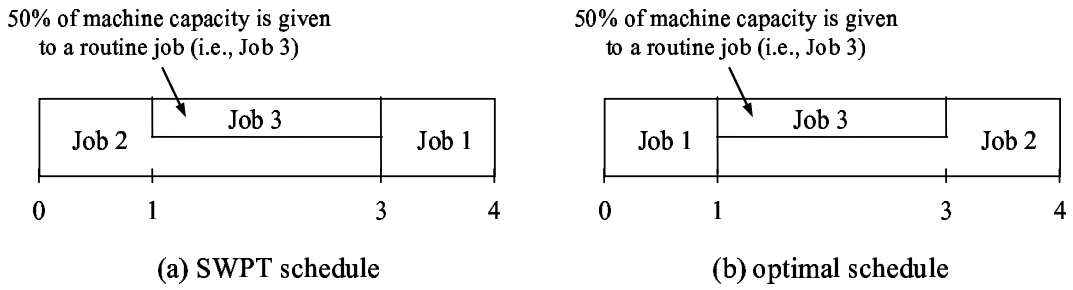


Figure 2: Example of Problem $1 \mid share(0.5) \mid \sum w_j C_j$.

Theorem 7 *For any rational number* $e \in (0,1)$*, the recognition version of problem* $1 \mid share(e) \mid \sum w_j C_j$ *is unary* NP*-complete.*

Proof. We show the *NP*-completeness of $1 \mid share(e) \mid \sum w_j C_j$ by reduction from 3-Partition, as defined in the proof of Theorem 1. Given an arbitrary instance of 3-Partition, we construct an instance of problem $1 \mid share(e) \mid \sum w_j C_j$ as follows:

$n = 4m$;

$\bar{n} = m$;

$w_j = p_j = e\hat{a}_j$, where $\hat{a}_j = M + a_j$ and $M = (2+e)mb$, for $j = 1, \ldots, 3m$;

$w_j = p_j = e(L - \hat{b}) + L$, where $\hat{b} = 3M + b$ and $L = m\hat{b} + 1$, for $j = 3m+1, \ldots, 4m$;

$p_j = (1-e)L$, for $j = 4m+1, \ldots, 5m$;

$r_j = 2(j - 4m - 1)L$, for $j = 4m+1, \ldots, 5m$;

$C = \frac{e}{2}[\sum_{j=1}^{3m} \hat{a}_j^2 + m^2\hat{b}^2 + m(m-1)\hat{b}(2L - \hat{b}) + 2m(m+1)(L - \hat{b})L] + m(m+1)L^2$.

Clearly, this construction requires polynomial time. Note that the w_j and p_j values in this constructed instance are rational numbers. They can be converted into integers through multiplying all these values by an appropriate integer. For simplicity, we ignore this conversion. We show that there exists a solution to this instance of problem $1 \mid share(e) \mid \sum w_j C_j$ with $\sum w_j C_j \leq C$ if and only if there exists a solution to the given instance of 3-Partition. In this construction, jobs $1, \ldots, 4m$ are primary jobs, whereas jobs $4m+1, \ldots, 5m$ are routine jobs. We refer to jobs $1, \ldots, 3m$ as "small jobs" and jobs $3m+1, \ldots, 4m$ as "large jobs."

($\Rightarrow$) Let $S_i = \{\pi_{3i-2}, \pi_{3i-1}, \pi_{3i}\}$, $i = 1, \ldots, m$, denote a solution to the given instance of 3-Partition, where $(\pi_1, \ldots, \pi_{3m})$ is a permutation of $(1, \ldots, 3m)$. Then, $a_{\pi_{3i-2}} + a_{\pi_{3i-1}} + a_{\pi_{3i}} = b$, $\hat{a}_{\pi_{3i-2}} + \hat{a}_{\pi_{3i-1}} + \hat{a}_{\pi_{3i}} = \hat{b}$, and $p_{\pi_{3i-2}} + p_{\pi_{3i-1}} + p_{\pi_{3i}} = e\hat{b}$, for $i = 1, \ldots, m$. Consider the schedule depicted in Figure 3. In this schedule, the processing of jobs $1, \ldots, 3m$ is shared

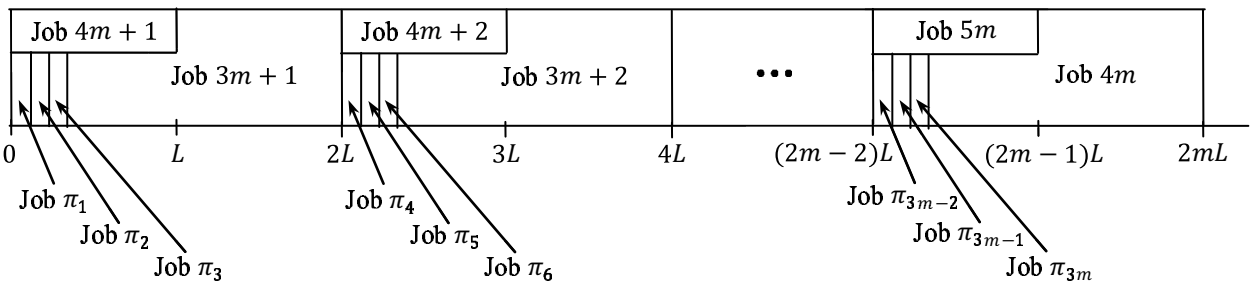
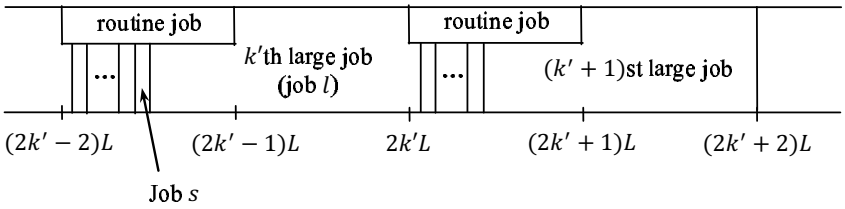


Figure 3: Schedule for the Given Instance of 3-Partition.

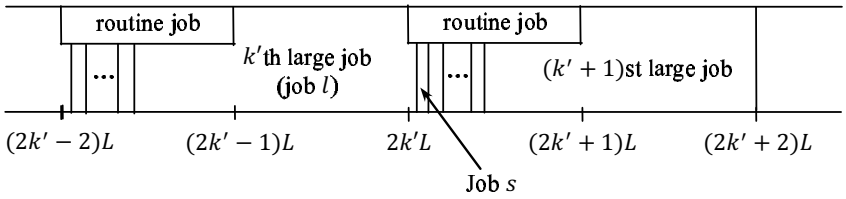
with the routine jobs. Thus, for $j = 1, \ldots, 3m$, job j requires $p_j/e = \hat{a}_j$ units of time to process. The processing of jobs $3m+1, \ldots, 4m$ is partially shared with the routine jobs. For $j = 3m+1, \ldots, 4m$, job j shares the processing capacity with a routine job during the time period $[2(j-1)L + \hat{b}, 2(j-1)L + L]$. During this period, $e(L - \hat{b})$ processing units of job j are completed. The remainder of job j is processed during the time period $[2(j-1)L + L, 2jL]$. Hence, for $i = 1, \ldots, m$, the completion times of small jobs π_{3i-2}, π_{3i-1}, and π_{3i} are $2(i-1)L + \hat{a}_{\pi_{3i-2}}$, $2(i-1)L + \hat{a}_{\pi_{3i-2}} + \hat{a}_{\pi_{3i-1}}$, and $2(i-1)L + \hat{a}_{\pi_{3i-2}} + \hat{a}_{\pi_{3i-1}} + \hat{a}_{\pi_{3i}}$, respectively, and the completion time of large job $3m + i$ is $2iL$. The total weighted completion time of jobs $1, \ldots, 4m$ is

$$
\begin{aligned}
&\sum_{i=1}^{m} \Big\{ e\hat{a}_{\pi_{3i-2}}[2(i-1)L + \hat{a}_{\pi_{3i-2}}] + e\hat{a}_{\pi_{3i-1}}[2(i-1)L + \hat{a}_{\pi_{3i-2}} + \hat{a}_{\pi_{3i-1}}] \\
&+ e\hat{a}_{\pi_{3i}}[2(i-1)L + \hat{a}_{\pi_{3i-2}} + \hat{a}_{\pi_{3i-1}} + \hat{a}_{\pi_{3i}}] + [e(L - \hat{b}) + L] \cdot 2iL \Big\} \\
&= e\sum_{i=1}^{m} \Big\{ \hat{a}_{\pi_{3i-2}}[(i-1)\hat{b} + \hat{a}_{\pi_{3i-2}}] + \hat{a}_{\pi_{3i-1}}[(i-1)\hat{b} + \hat{a}_{\pi_{3i-2}} + \hat{a}_{\pi_{3i-1}}] \\
&\quad + \hat{a}_{\pi_{3i}}[(i-1)\hat{b} + \hat{a}_{\pi_{3i-2}} + \hat{a}_{\pi_{3i-1}} + \hat{a}_{\pi_{3i}}] \Big\} \\
&\quad + e\sum_{i=1}^{m} (\hat{a}_{\pi_{3i-2}} + \hat{a}_{\pi_{3i-1}} + \hat{a}_{\pi_{3i}})[2(i-1)L - (i-1)\hat{b}] + [e(L - \hat{b}) + L] \sum_{i=1}^{m} 2iL \\
&= e\sum_{i=1}^{m} \Big[\hat{a}_{\pi_{3i-2}}(\hat{a}_{\pi_1} + \cdots + \hat{a}_{\pi_{3i-2}}) + \hat{a}_{\pi_{3i-1}}(\hat{a}_{\pi_1} + \cdots + \hat{a}_{\pi_{3i-1}}) + \hat{a}_{\pi_{3i}}(\hat{a}_{\pi_1} + \cdots + \hat{a}_{\pi_{3i}}) \Big] \\
&\quad + e\sum_{i=1}^{m} \hat{b}[2(i-1)L - (i-1)\hat{b}] + [e(L - \hat{b}) + L] \sum_{i=1}^{m} 2iL \\
&= \frac{e}{2}\sum_{j=1}^{3m} \hat{a}_j^2 + \frac{e}{2}\Big(\sum_{j=1}^{3m} \hat{a}_j\Big)^2 + \frac{e\hat{b}(2L - \hat{b})m(m-1)}{2} + [e(L - \hat{b}) + L]Lm(m+1) = C.
\end{aligned}
$$

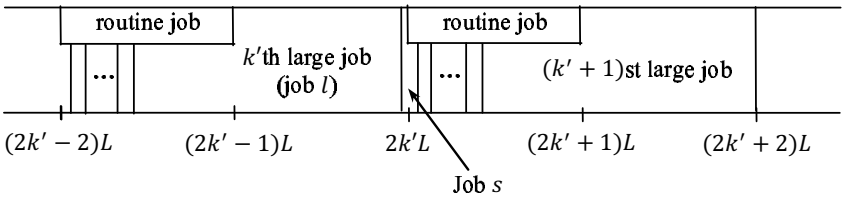
($\Leftarrow$) Suppose that there exists a feasible schedule for the above instance of problem $1 \mid share(e) \mid \sum w_j C_j$ with $\sum w_j C_j \leq C$. Then, there must exist a feasible schedule σ with



(a) Schedule σ



(b) Schedule σ_1 (Case 1.1)



(c) Schedule σ_1 (Case 1.2)

Figure 4: More Than $3k'$ Small Jobs Are Processed Before the k'th Large Job.

no inserted idle time for this instance such that $\sum w_j C_j \leq C$.

We first show that there exists a schedule σ' with no inserted idle time such that $\sum w_j C_j \leq C$ and that there are exactly $3k$ small jobs processed before the kth large job, for $k = 1, \ldots, m$; that is, there are three small jobs processed before the first large job and between any two consecutive large jobs. If there are exactly $3k$ small jobs processed before the kth large job for $k = 1, \ldots, m$ in schedule σ, then σ is the desired schedule σ'. Otherwise, let k' be the smallest value of k such that there are either more than or less than $3k$ small jobs processed before the kth large job. Let l denote the k'th large job in the schedule.

Case 1: More than $3k'$ small jobs are processed before job l. Then, job l must be completed after time $2k'L$ and before time $(2k'+1)L$. Let s be the small job processed immediately before job l (see Figure 4(a)). Denote the sum of weighted completion time of jobs s and l in this schedule as X_1. Then,

$$X_1 = e\hat{a}_s C_s(\sigma) + [e(L-\hat{b}) + L][C_s(\sigma) + 2L - \hat{b}],$$

where "$C_s(\sigma) + 2L - \hat{b}$" is the completion time of job l, since job l starts at time $C_s(\sigma)$, occupies 100% of the machine capacity for a period of length L, and occupies $e \times 100\%$ of the machine capacity for a period of length $L - \hat{b}$. Consider an alternative schedule σ_1 that is obtained by interchanging jobs s and l. Denote the sum of weighted completion time of jobs s and l in schedule σ_1 as X_2. We consider two different cases.

Case 1.1: After the job interchange, job l completes after time $2k'L$ (see Figure 4(b)). In this case, the completion times of jobs l and s are $C_s(\sigma) + 2L - \hat{b} - \hat{a}_s$ and $C_s(\sigma) + 2L - \hat{b}$, respectively, after the job interchange. Then,

$$X_2 = e\hat{a}_s[C_s(\sigma) + 2L - \hat{b}] + [e(L-\hat{b}) + L][C_s(\sigma) + 2L - \hat{b} - \hat{a}_s].$$

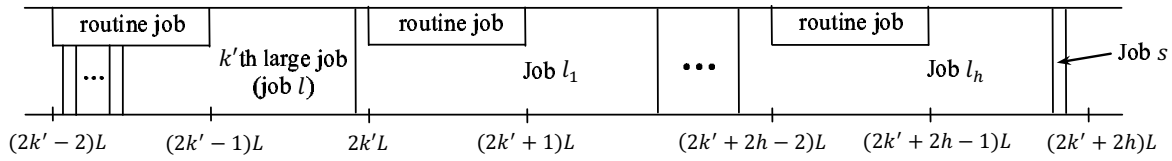
It is easy to verify that $X_2 < X_1$.

Case 1.2: After the job interchange, job l completes at or before time $2k'L$ (see Figure 4(c)). Let $2k'L - x$ be the completion time of job l after the job interchange, where $x \geq 0$. Since there are at least $3k'$ small jobs processed before job l and each of them has a processing time greater than eM, the total processing time of the small jobs, large jobs, and routine jobs completed by time $2k'L - x$ is greater than $3k'(eM) + k'[e(L-\hat{b}) + L] + k'(1-e)L = 2k'L - k'eb$, which implies that $x < k'eb < meb$. After the job interchange, the completion time of job s is $C_s(\sigma) + 2L - \hat{b}$, and the completion time of job l can be written as $C_s(\sigma) + 2L - \hat{b} - [x + \hat{a}_s - (x/e)]$, where "$x + \hat{a}_s - (x/e)$" is the duration of job s, since job s occupies 100% of the machine capacity for a period of length x and occupies $e \times 100\%$ of the machine capacity for a period of length $\hat{a}_s - (x/e)$. Thus,

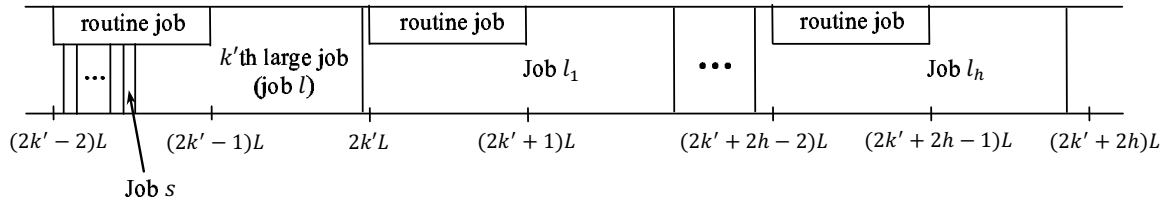
$$X_2 = e\hat{a}_s[C_s(\sigma) + 2L - \hat{b}] + [e(L-\hat{b}) + L]\Big\{C_s(\sigma) + 2L - \hat{b} - [x + \hat{a}_s - (x/e)]\Big\}.$$

Using the facts that "$x < meb$" and "$\hat{a}_s > (2+e)mb$," it is easy to verify that $X_2 < X_1$. Hence, in both Cases 1.1 and 1.2, schedule σ_1 has a smaller total weighted completion time than schedule σ.

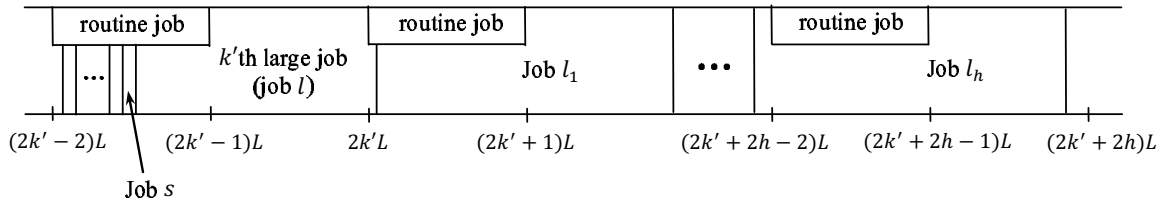
Case 2: Fewer than $3k'$ small jobs are processed before job l. Then, job l must be completed after time $(2k'-1)L$ and before time $2k'L$. Let s be the first small job processed after job l. Note that job s may be processed immediately after job l or after another large job. Let



(a) Schedule σ



(b) Schedule σ_2 (Case 2.1)



(c) Schedule σ_2 (Case 2.2)

Figure 5: Fewer Than $3k'$ Small Jobs Are Processed Before the k'th Large Job.

h be the number of large jobs scheduled between job l and job s, where $0 \leq h \leq m-1$ (see Figure 5(a)). For $i = 1, \ldots, h$, let l_i denote the ith large job processed after job l. Job l_i must be completed after time $(2k' + 2i - 1)L$ and before time $(2k' + 2i)L - p_s$. Denote the sum of weighted completion times of jobs $s, l, l_1, \ldots, l_h$ in this schedule as X_3. Then,

$$X_3 = e\hat{a}_s C_s(\sigma) + [e(L - \hat{b}) + L][C_l(\sigma) + C_{l_1}(\sigma) + \cdots + C_{l_h}(\sigma)].$$

Consider an alternative schedule σ_2 that is obtained by interchanging job s with jobs $l, l_1, \ldots, l_h$, i.e., inserting job s at the front of job l. Denote the sum of weighted completion times of jobs $s, l, l_1, \ldots, l_h$ in schedule σ_2 as X_4. Note that in schedule σ_2, the completion time of job l_i is $C_{l_i}(\sigma) + p_s$, for $i = 1, \ldots, h$. We consider two different cases.

Case 2.1: After the job interchange, job l completes at or before time $2k'L$ (see Figure 5(b)). In this case, after the job interchange, the completion time of job l is $C_l(\sigma) + p_s$. Thus,

$$X_4 = e\hat{a}_s C_s(\sigma_2) + [e(L - \hat{b}) + L]\Big\{[C_l(\sigma) + p_s] + [C_{l_1}(\sigma) + p_s] + \cdots + [C_{l_h}(\sigma) + p_s]\Big\}.$$

Note that $C_s(\sigma) = C_{l_h}(\sigma_2)$. Note also that in schedule σ_2, the duration of job l_i equals $2L - e\hat{b}$ for $i = 1, \ldots, h$, since job l_i occupies $e \times 100\%$ of the machine capacity for a period of length L and occupies 100% of the machine capacity for $L - e\hat{b}$ time units. The duration of job l is greater than $e(L - \hat{b}) + L$, since job l occupies only a portion of the machine capacity for some time period. Hence, $C_s(\sigma) - C_s(\sigma_2) = C_{l_h}(\sigma_2) - C_s(\sigma_2) > [e(L - \hat{b}) + L] + h(2L - e\hat{b})$. Using this inequality, it is easy to verify that $X_4 < X_3$.

Case 2.2: After the job interchange, job l completes after time $2k'L$ (see Figure 5(c)). Let $2k'L + y$ be the completion time of job l after the job interchange, where $y > 0$. Since there are at most $3k'$ small jobs processed before job l and their total processing time is at most $3k'eM + meb$, the total processing time of the small, large, and routine jobs completed by time $2k'L + y$ is at most $(3k'eM + meb) + k'[e(L - \hat{b}) + L] + k'(1 - e)L = 2k'L + (m - k')eb$, which implies that $y \leq (m - k')eb < meb$. Note that the completion time of job l in schedule σ_2 can be written as $C_l(\sigma) + e(\hat{a}_s - y) + y$. This is because moving job s in front of job l implies that $e\hat{a}_s$ units of processing time of job l need to be moved to the time interval $[C_l(\sigma), C_l(\sigma) + e(\hat{a}_s - y) + y]$, where $e(\hat{a}_s - y)$ units are processed with 100% machine capacity during the time interval $[C_l(\sigma), C_l(\sigma) + e(\hat{a}_s - y)]$, and ey units are processed with $e \times 100\%$ machine capacity during the time interval $[C_l(\sigma) + e(\hat{a}_s - y), C_l(\sigma) + e(\hat{a}_s - y) + y]$. Thus,

$$X_4 = e\hat{a}_s C_s(\sigma_2) + [e(L - \hat{b}) + L]\Big\{[C_l(\sigma) + e(\hat{a}_s - y) + y] + [C_{l_1}(\sigma) + p_s] + \cdots + [C_{l_h}(\sigma) + p_s]\Big\}.$$

Note that $C_s(\sigma) = C_{l_h}(\sigma_2)$. Note also that in schedule σ_2, the duration of job l equals $2L - e\hat{b}$ (since job l occupies 100% of the machine capacity for a period of length L and occupies $e \times 100\%$ of the machine capacity for a period of length $L - \hat{b}$). The duration of job l_i is at least $e(L - \hat{b}) + L$ for $i = 1, \ldots, h$. Hence, $C_s(\sigma) - C_s(\sigma_2) = C_{l_h}(\sigma_2) - C_s(\sigma_2) \geq (2L - e\hat{b}) + h[e(L - \hat{b}) + L]$. Thus,

$$\begin{aligned}
X_3 - X_4 &= e\hat{a}_s[C_s(\sigma) - C_s(\sigma_2)] - [e(L - \hat{b}) + L][e(\hat{a}_s - y) + y + hp_s] \\
&\geq e\hat{a}_s\Big\{(2L - e\hat{b}) + h[e(L - \hat{b}) + L]\Big\} - [e(L - \hat{b}) + L][e(\hat{a}_s - y) + y + he\hat{a}_s] \\
&= e\hat{a}_s(1 - e)L - (1 - e)[e(L - \hat{b}) + L]y \\
&> e(2 + e)mb(1 - e)L - (1 - e)[e(L - \hat{b}) + L]meb \\
&= e(1 - e)(L + e\hat{b})mb > 0,
\end{aligned}$$

where the second inequality follows from "$y < meb$" and "$\hat{a}_s > (2+e)mb$." Hence, $X_4 < X_3$. Therefore, in both Cases 2.1 and 2.2, schedule σ_2 has a smaller total weighted completion time than schedule σ. By repeatedly applying the job interchange argument presented in Cases 1 and 2, we conclude that there exists a feasible schedule σ' with no inserted idle time and $\sum w_j C_j \leq C$ such that there are three small jobs processed before the first large job and between any two consecutive large jobs.

Now, consider schedule σ'. Let (π_1, π_2, π_3) denote the sequence of small jobs processed before the first large job, and let $(\pi_{3i-2}, \pi_{3i-1}, \pi_{3i})$ denote the sequence of small jobs processed between the $(i-1)$st large job and the ith large job, for $i = 2, \ldots, m$. For $i = 1, \ldots, m$, let $u_i = \sum_{k=1}^{i}(\hat{a}_{\pi_{3k-2}} + \hat{a}_{\pi_{3k-1}} + \hat{a}_{\pi_{3k}}) - i\hat{b}$. It is easy to check that the start time of the ith large job is $2(i-1)L + \hat{b} + u_i$. Note that $u_m = 0$ and that $-ib < u_i < ib$ for $i = 1, \ldots, m-1$. For $i = 1, \ldots, m-1$, if $u_i < 0$, then the completion time of the ith large job is $2iL + eu_i$, which is less than $2iL$. In this case, the completion time of job π_{3i+1} is $2iL + u_i + \hat{a}_{\pi_{3i+1}}$, which is greater than $2iL$. For $i = 1, \ldots, m-1$, if $u_i \geq 0$, then the completion time of the ith large job is $2iL + u_i$, which is greater than or equal to $2iL$. In this case, the completion time of job π_{3i+1} is also $2iL + u_i + \hat{a}_{\pi_{3i+1}}$. Let $u_0 = 0$,

$$\phi_i = \begin{cases} eu_i, & \text{if } u_i < 0; \\ u_i, & \text{if } u_i \geq 0; \end{cases}$$

and

$$\Delta_i = [e(L - \hat{b}) + L]\phi_i - e(2L - \hat{b})u_i,$$

for $i = 1, \ldots, m$. Note that for $i = 1, \ldots, m$,

$$\sum_{k=1}^{3i-3} \hat{a}_{\pi_k} = u_{i-1} + (i-1)\hat{b} \tag{8}$$

and

$$\hat{a}_{\pi_{3i-2}} + \hat{a}_{\pi_{3i-1}} + \hat{a}_{\pi_{3i}} = u_i - u_{i-1} + \hat{b}. \tag{9}$$

Then, the total weighted completion time of jobs $1, \ldots, 4m$ is

$$\begin{aligned}
&\sum_{i=1}^{m} \Big\{ e\hat{a}_{\pi_{3i-2}}[2(i-1)L + u_{i-1} + \hat{a}_{\pi_{3i-2}}] + e\hat{a}_{\pi_{3i-1}}[2(i-1)L + u_{i-1} + \hat{a}_{\pi_{3i-2}} + \hat{a}_{\pi_{3i-1}}] \\
&+ e\hat{a}_{\pi_{3i}}[2(i-1)L + u_{i-1} + \hat{a}_{\pi_{3i-2}} + \hat{a}_{\pi_{3i-1}} + \hat{a}_{\pi_{3i}}] + [e(L - \hat{b}) + L](2iL + \phi_i) \Big\} \\
&= e\sum_{i=1}^{m} \Big\{ \hat{a}_{\pi_{3i-2}}[(i-1)\hat{b} + u_{i-1} + \hat{a}_{\pi_{3i-2}}] + \hat{a}_{\pi_{3i-1}}[(i-1)\hat{b} + u_{i-1} + \hat{a}_{\pi_{3i-2}} + \hat{a}_{\pi_{3i-1}}]
\end{aligned}$$

$$
\begin{aligned}
&+ \hat{a}_{\pi_{3i}}[(i-1)\hat{b} + u_{i-1} + \hat{a}_{\pi_{3i-2}} + \hat{a}_{\pi_{3i-1}} + \hat{a}_{\pi_{3i}}]\Big\} \\
&+ e\sum_{i=1}^{m}(\hat{a}_{\pi_{3i-2}} + \hat{a}_{\pi_{3i-1}} + \hat{a}_{\pi_{3i}})[2(i-1)L - (i-1)\hat{b}] + [e(L-\hat{b}) + L]\sum_{i=1}^{m}(2iL + \phi_i) \\
&= e\sum_{i=1}^{m}\Big[\hat{a}_{\pi_{3i-2}}(\hat{a}_{\pi_1} + \cdots + \hat{a}_{\pi_{3i-2}}) + \hat{a}_{\pi_{3i-1}}(\hat{a}_{\pi_1} + \cdots + \hat{a}_{\pi_{3i-1}}) + \hat{a}_{\pi_{3i}}(\hat{a}_{\pi_1} + \cdots + \hat{a}_{\pi_{3i}})\Big] \\
&+ e\sum_{i=1}^{m}(u_i - u_{i-1} + \hat{b})[2(i-1)L - (i-1)\hat{b}] + [e(L-\hat{b}) + L]\sum_{i=1}^{m}(2iL + \phi_i) \\
&= \frac{e}{2}\sum_{j=1}^{3m}\hat{a}_j^2 + \frac{e}{2}\Big(\sum_{j=1}^{3m}\hat{a}_j\Big)^2 + \frac{e\hat{b}(2L-\hat{b})m(m-1)}{2} + [e(L-\hat{b}) + L]Lm(m+1) \\
&+ e\sum_{i=1}^{m}(u_i - u_{i-1})[2(i-1)L - (i-1)\hat{b}] + [e(L-\hat{b}) + L]\sum_{i=1}^{m}\phi_i \\
&= C + e\sum_{i=1}^{m}(u_i - u_{i-1})[2(i-1)L - (i-1)\hat{b}] + [e(L-\hat{b}) + L]\sum_{i=1}^{m}\phi_i \\
&= C - e(2L-\hat{b})\sum_{i=1}^{m}u_i + [e(L-\hat{b}) + L]\sum_{i=1}^{m}\phi_i \\
&= C + \sum_{i=1}^{m}\Delta_i,
\end{aligned}
$$

where the second equality follows from equations (8) and (9). It is easy to check that $\Delta_i = 0$ if $u_i = 0$, and that $\Delta_i > 0$ if $u_i \neq 0$. Hence, $u_i = 0$ for all $i = 1, \ldots, m$, since otherwise the total weighted completion time of jobs $1, \ldots, 4m$ is greater than C. Let $S_i = \{\pi_{3i-2}, \pi_{3i-1}, \pi_{3i}\}$ for $i = 1, \ldots, m$. Then, $\{S_1, \ldots, S_m\}$ is a 3-partition of $\{1, \ldots, 3m\}$. □

3.2 Total completion time

We show that the SPT rule provides an optimal sequence for the total completion time objective.

Theorem 8 *For any $0 \leq e \leq 1$, problem $1 \mid share(e) \mid \sum C_j$ can be solved in $O(\tilde{n} + n\log n)$ time by scheduling the primary jobs in SPT order without inserted idle time.*

Proof. Since the objective is an increasing function of time, there exists an optimal schedule without inserted idle time. The remainder of the proof is by contradiction. Suppose there is an optimal schedule σ that includes a pair of adjacent primary jobs i and j, where i is processed before j, and $p_i > p_j$. We obtain a new schedule σ' from σ by interchanging jobs i and j. We then show that the total completion time of σ' is smaller than that of σ, contradicting the fact that σ is optimal.

Since the completion times of any job, other than i and j, are the same in both schedules, we compare the completion times of jobs i and j in both schedules. It is clear that $C_j(\sigma) = C_i(\sigma')$. To show that $C_i(\sigma) > C_j(\sigma')$, we begin at the start time of job i in schedule σ and trace the execution of job j. Since $p_j < p_i$, we have $C_j(\sigma') < C_i(\sigma)$, regardless of when the processing of jobs i and j is shared with the routine jobs. Hence, sequencing the primary jobs in SPT order yields an optimal schedule.

Sorting the primary jobs in SPT order requires $O(n \log n)$ time. Constructing the schedule for the primary and routine jobs requires $O(n + \tilde{n})$ time. Therefore, the overall computation time is $O(\tilde{n} + n \log n)$. □

We observe that the SPT order of the primary jobs is unaffected by the scheduled times of the routine jobs. Hence, even if there is uncertainty about those times, the algorithm we propose remains optimal.

3.3 Maximum lateness

For the $L_{\max}$ objective, sequencing the primary jobs in EDD order always provides the minimum value of $L_{\max}$, as the following result shows.

Theorem 9 *For any $0 \leq e \leq 1$, problem $1 \mid share(e) \mid L_{\max}$ can be solved in $O(\tilde{n} + n \log n)$ time by scheduling the primary jobs in EDD order without inserted idle time.*

Proof. Since the objective is a nondecreasing function of time, there exists an optimal schedule without inserted idle time. Suppose there is an optimal schedule σ that includes a pair of adjacent primary jobs i and j, where i is processed before j, and $d_i > d_j$. We obtain a new schedule σ' from σ by interchanging jobs i and j. We then show that the maximum lateness of σ' is not larger than the maximum lateness of σ.

It is clear that the completion times of every job, other than i and j, are identical in both σ and σ'. Thus, all we need to show is that $\max\{L_i(\sigma), L_j(\sigma)\} \geq \max\{L_i(\sigma'), L_j(\sigma')\}$. It is clear that $C_j(\sigma) = C_i(\sigma')$. Thus, $L_i(\sigma') = C_i(\sigma') - d_i < C_j(\sigma) - d_j = L_j(\sigma)$. Furthermore, $C_j(\sigma) > C_j(\sigma')$, and hence $L_j(\sigma) = C_j(\sigma) - d_j > C_j(\sigma') - d_j = L_j(\sigma')$. Thus, we have $L_j(\sigma) > \max\{L_i(\sigma'), L_j(\sigma')\}$, and hence $\max\{L_i(\sigma), L_j(\sigma)\} \geq \max\{L_i(\sigma'), L_j(\sigma')\}$. Repeating this interchange argument establishes the optimality of the EDD order. The analysis of the computation time follows that in Theorem 8. □

We observe that the EDD order of the primary jobs is unaffected by the scheduled times of the routine jobs. Hence, even if there is uncertainty about those times, the algorithm we propose remains optimal.

3.4 Number of late jobs

We consider the problem $1 \mid share(e) \mid \sum U_j$. Our work in this section extends that of [16] for the classical problem $1 \mid\mid \sum U_j$. We first state a property of an optimal schedule. This result is similar to Lemma 1 in [8].

Lemma 4 *For any $0 \leq e \leq 1$, there exists an optimal schedule for problem $1 \mid share(e) \mid \sum U_j$ in which all the on-time primary jobs are sequenced by the EDD rule, followed by all late primary jobs in arbitrary order, with no inserted idle time.*

Proof. For a given schedule σ, $U_j(\sigma)$ is a nondecreasing function of $C_j(\sigma)$. Therefore, there exists an optimal schedule with no inserted idle time. For a given instance of the problem, let E and T denote the sets of on-time and late primary jobs, respectively. A pairwise interchange argument shows that there exists an optimal schedule where all the jobs of E are scheduled before any job of T. Since the jobs of T do not affect the objective, we assume that they are scheduled in arbitrary order. It remains to sequence the jobs of E. To ensure that the maximum lateness of the jobs of E is zero or negative, it suffices to sequence the jobs of E to minimize their maximum lateness. Therefore, from Theorem 9, there exists an optimal schedule where the jobs of E are sequenced by the EDD rule. □

Algorithm ShareU

1. Sort the primary jobs in nondecreasing order of their due dates; i.e., $d_1 \leq d_2 \leq \cdots \leq d_n$.
2. $i := 1$; $E := \emptyset$; $T := \emptyset$; $t := 0$.
3. Add primary job i to set E, assign it to start at time t, and let t' denote the time at which it completes. Note that any routine jobs which are processed between times t and t' share processing with job i.
4. If $t' > d_i$, then remove the longest job from E and the current schedule, add it to T, eliminate all idle time between the primary jobs, and let t' be the completion time of the last primary job in the resulting schedule.

5. If $i = n$, then schedule the jobs in T at the end of the schedule, and stop. Otherwise, $i := i + 1$; $t := t'$; and go to Step 3.

For any primary job subset $S \subseteq \{1, 2, \ldots, n\}$, let $\ell(S)$ denote the completion time of the last job in S if we schedule the jobs in S in EDD order in front of the jobs in $\{1, 2, \ldots, n\} \setminus S$, with no idle time between jobs. The optimality of Algorithm ShareU is based on the following property.

Lemma 5 *For an instance of problem $1 \mid share(e) \mid \sum U_j$, consider the subset of primary jobs S. Let k and k' be any two jobs in S. If $p_k \leq p_{k'}$, then $\ell(S \setminus \{k\}) \geq \ell(S \setminus \{k'\})$.*

Proof. By contradiction, suppose $\ell(S \setminus \{k\}) < \ell(S \setminus \{k'\})$. Then, let x and y denote the total amount of processing of the routine jobs during the time intervals $[\ell(S \setminus \{k\}), \ell(S \setminus \{k'\})]$ and $[\ell(S \setminus \{k'\}), \ell(S)]$, respectively.

We first consider the case where $e > 0$. We have

$$x \leq (1 - e)[\ell(S \setminus \{k'\}) - \ell(S \setminus \{k\})], \tag{10}$$

$$\ell(S) - \ell(S \setminus \{k'\}) = p_{k'} + y, \tag{11}$$

and

$$\ell(S) - \ell(S \setminus \{k\}) = p_k + x + y. \tag{12}$$

From (10)–(12), and since $p_k \leq p_{k'}$, we have

$$\ell(S) - \ell(S \setminus \{k\}) \leq [\ell(S) - \ell(S \setminus \{k'\})] + (1 - e)[\ell(S \setminus \{k'\}) - \ell(S \setminus \{k\})].$$

This implies that

$$e \cdot \ell(S \setminus \{k'\}) \leq e \cdot \ell(S \setminus \{k\}),$$

which contradicts that $\ell(S \setminus \{k\}) < \ell(S \setminus \{k'\})$.

Next, we consider the case where $e = 0$. In this case,

$$x < \ell(S \setminus \{k'\}) - \ell(S \setminus \{k\}), \tag{13}$$

where the strict inequality follows from the fact that part of the time interval $[\ell(S \setminus \{k\}), \ell(S \setminus \{k'\})]$ must be spent on the processing of some primary job(s). Note that in this case

equations (11) and (12) remain valid. However, conditions (11)–(13) imply that $p_k > p_{k'}$, which is a contradiction. □

Lemma 5 implies that if primary job sets S and S' differ from each other by only one job, for example $S \setminus S' = \{k\}$ and $S' \setminus S = \{k'\}$, and if $p_{k'} \leq p_k$, then the completion time of the last job in S', when the jobs are processed in EDD order, is no greater than the completion time of the last job in S, when the jobs are processed in EDD order. We now present the main result of this section.

Theorem 10 *For any $0 \leq e \leq 1$, Algorithm ShareU finds an optimal schedule for problem $1 \mid share(e) \mid \sum U_j$ in $O(\tilde{n} + n\log(\tilde{n}n))$ time.*

Proof. The optimality of Algorithm ShareU can be proved by applying Lemmas 4 and 5 and applying a similar argument to that in the proof of Theorem 4 in [8]. The details of the optimality proof are omitted.

We consider the running time of Algorithm ShareU. The running time is dominated by the loop in Steps 3–5, which has n iterations. In order to compute t' efficiently in each iteration, we precompute two arrays $A[\cdot]$ and $B[\cdot]$. We use $A[2i-1]$ and $A[2i]$ to store the start time and finish time, respectively, of routine job $n+i$ for $i = 1, \ldots, \tilde{n}$. We use $B[k]$ to store the total amount of processing available for primary jobs between time 0 and time $A[k]$, for $k = 1, \ldots, 2\tilde{n}$. Thus, $B[2i-1] = B[2i-2] + (A[2i-1] - A[2i-2])$ and $B[2i] = B[2i-1] + e(A[2i] - A[2i-1])$ for $i = 1, \ldots, \tilde{n}$, where $A[0] = B[0] = 0$. Given any set E of primary jobs and their total processing requirement P, we obtain t', i.e., the completion time of the last job in E, in $O(\log \tilde{n})$ time via a binary search of array $B[\cdot]$. If $P \in [B[2l-1], B[2l])$, then $t' = A[2l-1] + (P - B[2l-1])/e$. If $P \in [B[2l], B[2l+1])$, then $t' = A[2l] + (P - B[2l])$. In Step 4, if $t' > d_i$, then we need to determine the longest job in E. If we maintain the jobs in E as a heap, it takes $O(\log n)$ time to insert a single job or delete a longest job. Hence, when arrays $A[\cdot]$ and $B[\cdot]$ are precomputed, the total running time of the loop is $O(n(\log \tilde{n} + \log n)) = O(n\log(\tilde{n}n))$. Precomputing arrays $A[\cdot]$ and $B[\cdot]$ requires $O(\tilde{n})$ time. Therefore, the overall computational requirement of Algorithm ShareU is $O(\tilde{n} + n\log(\tilde{n}n))$ time. □

4 Concluding Remarks

This paper studies two multitasking system designs for two different types of schedule disruption that arise in practice, and describes how the resulting mathematical models can be solved. For each system design, we consider four of the most important practical scheduling objectives, and for each, where possible, describe an efficient optimal algorithm. We also provide intractability results that define limits on the solvability of some of the problems.

Table 1 summarizes our algorithm and complexity results. The table provides the running time of all our polynomial and pseudo-polynomial time algorithms. Also, we use "BNPC" (respectively, "UNPC") to denote that the recognition version of a problem is binary (respectively, unary) *NP*-complete. Each cell in the table contains a reference to a location in the paper where the corresponding result can be found.

	$\sum w_j C_j$	$\sum C_j$	$L_{\max}$	$\sum U_j$
Alternate Period Processing	UNPC Thm. 1	$O(nP)$, BNPC Thm. 2, Thm. 3	$O(nP)$, BNPC Thm. 4, Thm. 5	$O(n^2P)$, BNPC Thm. 6, Cor. 1
Shared Processing with Routine Activities	UNPC Thm. 7	$O(\tilde{n} + n \log n)$ Thm. 8	$O(\tilde{n} + n \log n)$ Thm. 9	$O(\tilde{n} + n \log(\tilde{n}n))$ Thm. 10

Table 1: Summary of Algorithm and Complexity Results.

We make the following observations. The alternative period processing model is a simple and practical design to allow workers to rest and prevent them from becoming bored. However, using this model, the scheduling problems become less computationally tractable than the corresponding classical problems. On the other hand, the shared processing model for handling routine activities is computationally tractable when the objective is $\sum C_j$, $L_{\max}$, or $\sum U_j$. In fact, when the objective is $\sum C_j$ or $L_{\max}$, a simple sequencing rule for the primary jobs generates optimal solutions.

A number of interesting problems remain open for further research. First, the classical scheduling literature contains a large number of problems that remain to be studied under the two types of disruptions discussed here. Second, our work motivates the development and evaluation of practical measures for handling disruptions. Finally, our work motivates research to develop and analyze scheduling models with multitasking features, which is an important research direction due to the prevalence of multitasking in business applications.

Acknowledgments

The authors thank the three anonymous referees for their helpful comments and suggestions. This work is supported in part by the Summer Fellowship Program, Fisher College of Business, The Ohio State University, to the first author; in part by the National Science Foundation under grant CMMI-0969830, to the second author; and in part by Research Grants Council of Hong Kong under grant PolyU5195/13E, to the third author.

References

[1] G. Appasami, K. Suresh Joseph, Optimization of operating systems towards green computing, *International Journal of Combinatorial Optimization Problems and Informatics* 2(3) (2011) 39–51.

[2] B. Detienne, A mixed integer linear programming approach to minimize the number of late jobs with and without machine availability constraints, *European Journal of Operational Research* 235 (2014) 540–552.

[3] B.T. Doshi, Queueing systems with vacations—A survey, *Queueing Systems* 1 (1986) 29–66.

[4] J. Du, J.Y-T. Leung, Minimizing total tardiness on one machine is NP-hard, *Mathematics of Operations Research* 15 (1990) 483–495.

[5] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* Freeman, New York, 1979.

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

[7] R.R. Guadaña, M.R. Perez, L. Rutaquio, Jr., A comprehensive review for central processing unit scheduling algorithm, *International Journal of Computer Science Issues* 10(1) (2013) 353–358.

[8] N.G. Hall, J.Y.-T. Leung, C.-L. Li, The effects of multitasking on operations scheduling, *Production and Operations Management* 24 (2015) 1248–1265.

[9] N. Hashemian, C. Diallo, B. Vizvári, Makespan minimization for parallel machines scheduling with multiple availability constraints, *Annals of Operations Research* 213 (2014) 173–186.

[10] W.A. Horn, Minimizing average flow time with parallel machines, *Operations Research* 21 (1973) 846–847.

[11] Y. Huo, H. Zhao, Total completion time minimization on multiple machines subject to machine availability and makespan constraints, *European Journal of Operational Research* 243 (2015) 547–554.

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

[13] J. Kaabi, Y. Harrath, A survey of parallel machine scheduling under availability constraints, *International Journal of Computer and Information Technology* 3 (2014) 238–245.

[14] I. Kacem, H. Kellerer, Y. Lanuel, Approximation algorithms for maximizing the weighted number of early jobs on a single machine with non-availability intervals, *Journal of Combinatorial Optimization* 30 (2015) 403–412.

[15] Y. Ma, C. Chu, C. Zuo, A survey of scheduling with deterministic machine availability constraints, *Computers & Industrial Engineering* 58 (2010) 199–211.

[16] J.M. Moore, An n job, one machine sequencing algorithm for minimizing the number of late jobs, *Management Science* 15 (1968) 102–109.

[17] J. Noguera, R.M. Badia, Multitasking on reconfigurable architectures: Microarchitecture support and dynamic scheduling, *ACM Transactions on Embedded Computing Systems* 3 (2004) 385–406.

[18] M.L. Pinedo, *Scheduling: Theory, Algorithms, and Systems,* 4th edition, Springer, New York, 2012.

[19] S. Sachdeva, P. Panwar, A review of multiprocessor directed acyclic graph (DAG) scheduling algorithms, *International Journal of Computer Science & Communication* 6(1) (2015) 67–72.

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

[21] C. Steiger, H. Walder, M. Platzner, Operating systems for reconfigurable embedded platforms: Online scheduling of real-time tasks, *IEEE Transactions on Computers* 53 (2004) 1393–1407.

[22] J. Teghem, Jr., Control of the service process in a queueing system, *European Journal of Operational Research* 23 (1986) 141–158.

[23] G. Wang, H. Sun, C. Chu, Preemptive scheduling with availability constraints to minimize total weighted completion times, *Annals of Operations Research* 133 (2005) 183–192.

[24] W. Wang, S. Ranka, P. Mishra, Energy-aware dynamic slack allocation for real-time multitasking systems, *Sustainable Computing: Informatics and Systems* 2 (2012) 128–137.

[25] X. Wang, T.C.E. Cheng, A heuristic for scheduling jobs on two identical parallel machines with a machine availability constraint, *International Journal of Production Economics* 161 (2015) 74–82.