The Effects of Multitasking on Operations Scheduling

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

Abstract

This paper considers a typical scheduling environment that is influenced by the behavioral phenomenon of multitasking. Under multitasking, the processing of a selected job suffers from interruption by other jobs that are available but unfinished. This situation arises in a wide variety of applications; for example, administration, manufacturing, and process and project management. Several classical solution methods for scheduling problems no longer apply in the presence of multitasking. The solvability of any scheduling problem under multitasking is no easier than that of the corresponding classical problem. We develop optimal algorithms for some fundamental and practical single machine scheduling problems with multitasking. For other problems, we show that they are computationally intractable, even though in some cases the corresponding problem in classical scheduling is efficiently solvable. We also study the cost increase and value gained due to multitasking. This analysis informs companies about how much it would be worthwhile to invest in measures to reduce or encourage multitasking.

Keywords: scheduling, multitasking, polynomial time algorithm, cost and value of multitasking.

1 Introduction

The behavioral phenomenon known as *multitasking* can be observed numerous times each day. Multitasking is defined as "The performance of multiple tasks at one time" (Merriam-Webster Online 2014). So widespread is this phenomenon that Salvucci and Taatgen (2011) comment, "In some cases, it almost seems that people have a compelling need for multitasking . . ."

Several research studies suggest that multitasking is both frequent and costly. O'Leary et al. (2006) find that hospitalists spend 21% of their time working on more than one activity. G. Weinberg, as quoted in theSalmonFarm (2007), estimates that adding a second task results in a 20% loss of productivity, and adding a third task increases this loss to almost 50%. Järrehult (2012) uses a numerical example to show how multitasking can increase project time by over 60%. A case study (Realization 2014) of 45 organizations with multitasking reduction programs estimates the median improvement in productivity at 59.8%. Extrapolating these results globally suggests an annual cost of $450 billion from multitasking. Also, Suddath (2012) quotes the IT research and consulting firm Basex as estimating the annual cost of multitasking at $1 trillion globally.

An early criticism of human multitasking is offered by Philip Stanhope, 4th Earl of Chesterfield (1694–1773), in a 1740s letter to his son, "There is enough time for everything in the course of the day, if you do but one thing at once, but there is not enough time in the year, if you will do two things at a time" (Stanhope 1847). More recently, multitasking in personal interactions and in technology applications is the subject of frequent social commentary (Rosen 2008). This paper, however, studies *the effect of multitasking on the efficiency of administrative and manufacturing systems, and process management applications*. An important example of the last, where multitasking is frequently observed, is project management (Klastorin 2004, Kerzner 2013).

Discussions of the motivations for human multitasking can be found in the literature of behavioral psychology, operations management, cognitive engineering, and project management. Jez (2011) provides an integrative review of the multitasking literature. We identify five principal motivations for multitasking in administrative and business processes:

1. A need to feel or appear productive.
2. A need to demonstrate progress on different tasks or treat task owners equitably.
3. Anxiety about the processing requirements of waiting tasks.

4. A need for variety in work.
5. Interruption by routine scheduled activities.

In the following paragraphs, we discuss each of these motivations for multitasking in turn, and provide some supporting references.

An important motivation for multitasking is a need to feel productive, or to appear to be productive to one's co-workers. Huff (2007) quotes an interview subject about multitasking, "I enjoy it. I like the feeling of accomplishment I get when I can do two or three things to completion at the same time." She attributes the frequency of multitasking to pride. Damian (2009) reports a psychology study which indicates that many people believe, albeit sometimes erroneously, that they are good at multitasking. Cantor (2010) reports that many job listings identify an ability to multitask effectively as an important job attribute. Hence, once hired, employees multitask in anticipation of being appreciated for doing so.

The need to report progress on different tasks or to treat task owners equitably can be a powerful motivation for multitasking, especially in project management applications. Elder (2006) discusses situations where each task has an owner, and the owner typically makes frequent inquiries about progress on the task. Multitasking is an immediate consequence of such business pressures (Morgenstern 2004). Rand (2000) describes multitasking as an important aspect of project management that needs to be controlled. Leach (1999) discusses the prevalence and negative consequences of multitasking in project management.

Another significant motivation for multitasking is anxiety about the processing requirements of the waiting tasks. One source of anxiety is uncertainty about when the waiting tasks will be scheduled (Morgenstern 2004). This anxiety can sometimes be mitigated by the development of a detailed schedule. A second source of anxiety is concern about a lack of information (Morgenstern 2004). Relevant information that may be missing includes how difficult, or how time consuming, the waiting tasks will turn out to be once they are started and more information on them becomes available. Hence, multitasking of one or more of the waiting tasks in effect serves as a sampling procedure that may mitigate this type of anxiety.

Regarding the need for variety, boredom while working on a single task is a major factor in motivating multitasking. The blog Get More YouTube Views (2010) comments "Most people have an attention span of around 10 minutes. This means that after 10 minutes they will start feeling

bored, annoyed, angry or anxious and will automatically switch to something different in order to relieve the stress." What they switch to is typically a different task that is waiting. Cantor (2010) provides a similar perspective, "Multitasking is the art of distracting yourself from two things you'd rather not be doing by doing them simultaneously." She also identifies boredom as one of the major causes of multitasking.

Finally, many business processes rely on routine activity that interrupts a continuing schedule. Examples include spending the first part of the day answering e-mail, or performing end-of-day file backup and maintenance. As a result of such activities, a primary task that is scheduled over several days is necessarily interrupted. Jett and George (2003) study the consequences of interruptions, including scheduled breaks, for organizational activities. At a more operational level, Fuhrmann and Cooper (1985) study queueing systems with random job arrivals and generalized vacations, which represent time set aside for other activities. Lee (1996) studies a scheduling system where machines are unavailable due to preventive maintenance. Ma et al. (2010) provide a survey of 85 papers on the closely related topic of scheduling with known machine unavailability times.

Under the five motivations discussed above, multitasking results from a lack of focus on the currently scheduled work, relative to alternative available work. In none of those cases is multitasking the result of uncertainty in the planning environment. For this reason, our paper models the effect of multitasking *deterministically*. There are other potential motivations for multitasking, for example the random arrival of urgent work (Pinedo 2012), that require stochastic modeling. Those motivations are not studied as part of the current paper.

Our work is distinguished from classical scheduling in five ways. First, in classical scheduling, no work is ever processed except as a result of a deliberate decision by a decision maker. However, in all the above situations, multitasking is inadvertent. Second, in some situations inadvertent multitasking can add cost, but not necessarily value, to a schedule. However, the psychological benefit of satisfying "a need to feel or appear productive" or "a need for variety in work," for example, can add value to the process by speeding up work. In such situations, multitasking can generate value as well as improve quality of work (Loeb and Alluisi 1977, Ware and Baker 1977, Craig 1985). Third, in classical preemptive scheduling (Pinedo 2012), a preempting job is typically processed to completion before resumption of a job which it has preempted, whereas under multitasking an interrupting job is only partially processed. Fourth, in classical processor sharing

models (Gonzalez 1977), jobs are processed simultaneously, whereas under multitasking there is a transition from one job to another with a switching time. Further, classical processor sharing is deliberate, whereas multitasking is inadvertent. Fifth, most algorithms for classical preemptive scheduling fail to deliver optimal solutions in multitasking situations.

This paper evaluates the effects of multitasking in a simple task scheduling environment. This environment has been extensively studied within the classical scheduling literature, leading to the development of simple indexing rules and more complex optimal and heuristic algorithms. However, most of these classical results no longer apply in the presence of multitasking. We develop new solution procedures for several of the most widely studied classical scheduling problems, under multitasking. We also consider the extent by which multitasking increases cost or adds value. This analysis informs companies about how much it would be worthwhile to invest in measures to eliminate multitasking.

The remainder of the paper is organized as follows. Section 2 describes a small, practical case study that illustrates the application of scheduling with multitasking. In Section 3, we describe our notation, formally define the problems to be studied, develop a mathematical model of multitasking, and provide a general observation about problem solvability. Sections 4, 5, 6, and 7 discuss the solvability of problems with the total weighted completion time objective, the maximum lateness objective, the number of late jobs objective, and the weighted number of late jobs objective, respectively. Section 8 computationally evaluates the increase in cost and value gained due to multitasking for these four problems. Finally, Section 9 provides a conclusion and an extensive list of suggestions for future research.

2 Case Study

We describe a practical administrative planning scenario that illustrates a scheduling problem with multitasking.

Planning Scenario

An administrator provides support for the marketing and promotional activities of various departments at the university where she works. Her major tasks include developing promotional brochures

for academic programs, promoting major events, and writing newsletters and annual reports. She works with a small team of junior staff, who provide her with support, including collection of information, liaison with the other departments in the university, and preliminary drafting. To develop a set of promotional materials, her team members spend their time preparing the materials and then pass the prepared materials to her for finalization. During this preparation process, she needs to communicate and meet with the team members regularly so that they deliver a quality product. Each major task has a due date at which the materials will be published. The administrator's performance is based on meeting her due dates.

Need for Multitasking

The administrator normally works on one major task at a time. However, while doing so, she also needs to spend some time on each unfinished (i.e., waiting) task communicating with her team. To implement this, every time she switches from one primary task to another, she gives some time to review the team's progress of all the waiting tasks and provide feedback to them.

Data Requirements

(i) Each major task requires between 10 and 60 work hours, with quite predictable duration based on the tasks' repetitive nature and the administrator's past experience with similar tasks.

(ii) Every time the administrator reviews the waiting tasks, she spends about 0.25 hours discussing the progress of each waiting task with her team members, then spends another 2 hours working on each waiting task by herself.

(iii) However, if most of the waiting task is already completed, she does not spend additional time working on it until that task is ready for her to finalize. As a rule of thumb, once she has spent 50% of her time on a waiting task, she stops working on it herself, but spending 0.25 hours reviewing the task with the team is still necessary.

In Section 3, following the multitasking literature, we describe the meeting time of 0.25 hours per waiting task as a "switching time," and the 2 hours of working time as an "interruption time," and we also provide notation for the above data requirements.

3 Model Description and Properties

Our scheduling environment with multitasking can be described as follows. There is a given set of tasks that need to be processed by a work center. When a task is being processed, it is interrupted by other unfinished tasks that are available for processing. In order to present this model using classical machine scheduling terminology, we refer to the work center as a "machine" and the tasks as "jobs."

We let $N = \{1, 2, \ldots, n\}$ denote a given set of jobs. Job j has a processing time $p_j > 0$, and in some of the problems we consider also a due date $d_j \geq 0$ and a weight $w_j > 0$, for $j = 1, 2, \ldots, n$. We let $P = \sum_{j=1}^{n} p_j$ and $W = \sum_{j=1}^{n} w_j$. For simplicity, we assume that p_j and w_j are integer-valued. A single machine is available for processing the jobs, and all jobs are available for processing at time 0. The machine can process only one job at a time. Whatever job is scheduled at any point in time is denoted as the *primary job*. We refer to jobs that are available, and not yet fully processed, as *waiting jobs*. Under multitasking, when a job is being processed, it is unavoidably interrupted by the waiting jobs. We let $S_j \subset N$ denote the set of jobs that are waiting when job j is the primary job. We note that S_j may include jobs that have already been partially processed while interrupting other jobs. The partial processing of a job that occurs while it interrupts other jobs need not be repeated. Preemption of a primary job is not allowed, except as a result of multitasking. Hence, a primary job is always completed before another job becomes the primary job. Thus, every job needs to be selected as the primary job exactly once. Therefore, the only decision that needs to be made is a job sequence σ in which the jobs of N appear as primary jobs.

In order to model the process by which the waiting jobs interrupt the primary jobs, we introduce two assumptions. The first assumption is that the time during which the waiting job interrupts the primary job, which we refer to as the *interruption time*, is independent of the characteristics of the primary job. The second assumption is that the amount of time during which no job is being processed while the primary job is scheduled, which we refer to as the *switching time*, depends only on the number of waiting jobs. Both these assumptions are justified from the empirical literature of multitasking in the remainder of this section.

Let $i \in S_j$ be a waiting job when j is scheduled as a primary job. Let p'_i denote the remaining processing time of job i at the start of a period when job j is scheduled as a primary job. The

amount of time for which job i interrupts job j, i.e., the interruption time, is given by $g_i(p'_i)$, where $g_i(\cdot)$ is an arbitrary nonnegative function such that $0 \leq g_i(p'_i) < p'_i$. As mentioned above, this portion of processing of job i need not be repeated when job i becomes a primary job. When primary job j is being processed, it also incurs an overhead $f(|S_j|)$, i.e., the amount of switching time for handling the interrupting jobs, where $f(0) = 0$. Since during this switching time no useful work is performed, the makespan of the schedule is increased by $f(|S_j|)$ for each primary job j, and hence is longer than the total processing time of all the jobs. Thus, the total amount of interruption during the processing of job j is $f(|S_j|) + \sum_{i \in S_j} g_i(p'_i)$. This *multitasking function* is flexible enough to scale its two components relative to each other as required. We refer to $f(\cdot)$ and $g_i(\cdot)$ as the *switching function* and *interruption function*, respectively.

We allow $f(|S_j|)$ to be positive, zero, or negative. A negative switching time represents the situation where the value of interruption dominates the overhead. With respect to the function $g_i(p'_i)$, we define $h_i(l)$, $0 \leq l \leq n-1$, to be the remaining processing time of job i after it has interrupted l primary jobs. Thus, $h_i(0) = p_i$, $h_i(1) = p_i - g_i(p_i)$, $h_i(2) = h_i(1) - g_i(h_i(1))$, $\ldots$, $h_i(n-1) = h_i(n-2) - g_i(h_i(n-2))$. We assume that $g_i(\cdot)$ is defined in such a way that $h_i(l) + f(k)$ is a positive value for all $0 \leq l \leq n-1$ and $k \geq 0$. This assumption ensures that the remaining processing time of job i is positive when it becomes a primary job and when the switching time is included.

Returning to the case study described in Section 2, we are now able to specify a switching function $f(|S_j|) = (0.25)|S_j|$, where each time unit is 1 hour, and an interruption function $g_i(p'_i) = 2$ if $p'_i \geq (0.5)p_i$, and $g_i(p'_i) = 0$ otherwise.

We now justify our multitasking function directly from the empirical literature of multitasking (Chisholm et al. 2000, Rubinstein et al. 2001, Czerwinski et al. 2004, González and Mark 2004, Altmann and Trafton 2007, Iqbal and Horvitz 2007). This literature is grounded in information technology and health care applications, both of which introduce application-specific factors. Within information technology applications, multitasking functions are influenced by the relative levels of cognitive complexity, for example the amount of creativity and reasoning required, of the primary and waiting jobs. Within health care applications, multitasking functions are influenced by distraction during interruption, for example when the interrupting task is itself interrupted. Since our objective is to model generic applications that are not specific to a particular industry,

we do not model these application-specific factors.

However, the empirical literature of multitasking also identifies two factors that potentially apply across various applications. The first factor is the amount of the task remaining to complete (Iqbal and Bailey 2006), which we operationalize as the remaining processing time of the primary task at the time when interruption occurs. The second factor is the potential distraction during interruption (Grundgeiger et al. 2010), which we operationalize as the remaining processing time of the interrupting task. The above two works apply these factors as independent variables in multiple linear regression models to predict the *resumption lag*, which is the time between the end of processing of the interrupting jobs and the resumption of processing of the primary task. These factors are used in the design of our above multitasking function.

Regarding the statistical significance of the remaining processing time of the primary job in predicting interruption time, we make use of two empirical results. First, the length of the interruption has a very strong correlation with the length of the resumption lag (Grundgeiger et al. 2010). Second, the remaining processing time of the primary job is not a significant predictor of the length of the resumption lag (Iqbal and Bailey 2006). Combining these two results, we infer that the remaining processing time of the primary job is not a significant predictor of the length of interruption. This conclusion is supported by Iqbal and Bailey (2006), who comment that the cost of interruption "depends more on the characteristics that reflect current and prospective allocation of mental resources (workload) than on those that reflect temporal position . . ."

Recall that p'_i is the remaining processing time of waiting job i at the moment when job j becomes a primary job. Various assumptions are possible about the relationship between p'_i and the interruption time. For example, in classical preemptive scheduling (Pinedo 2012), the interrupting job is processed to completion. Alternatively, in round-robin scheduling (Coffman et al. 1970), the interrupting job is processed for a fixed amount of time that is independent of the characteristics of that job. The empirical literature of multitasking provides some related results. Grundgeiger et al. (2010) identify a very strong statistical relationship both between the amount of interruption and the resumption lag, and between the amount of potential distraction and the resumption lag, where the p-value is < 0.001 in both cases. Combining these two results, we infer a very strong statistical relationship between the amount of potential distraction and the interruption time. In the generic applications we consider, the amount of potential distraction is represented by the

remaining processing time of the interrupting task, p'_i. Therefore, we use the remaining processing time of the interrupting task as a major predictor of interruption time, and define $g_i(\cdot)$ as a function of p'_i.

Recall that we have defined the switching time as an unproductive time during which no job is being processed. The existence of switching time is well recognized in the literature (Speier et al. 1999, Rubinstein et al. 2001, Seshadri and Shapira 2001, Dobson et al. 2013). In a generic application, switching time is essentially a "fixed setup time" for the unloading or shutdown and the loading or startup of a job. Hence, our multitasking function models the total switching time $f(|S_j|)$ during the processing of primary job j as depending only on the number of switches incurred during that processing, one for each waiting job. As discussed above, in some applications, multitasking can add value to the process. For example, when workers need to demonstrate progress on different tasks or to treat task owners equitably, putting the primary job on hold temporarily and processing part of a waiting task can increase worker satisfaction and processing efficiency. We model this "value of interruption" as a reduction in switching time. That is, our switching time equals the shutdown/startup time of a job minus the value of interruption obtained from job switching. We permit the switching time to be negative. This allows the possibility that the value of interruption exceeds the shutdown/startup time of a job.

In view of this discussion, our above multitasking function depends on both the number and the remaining processing times of the interrupting jobs, but not on the remaining processing time of the primary job. Further, both switching time and interruption time are general functions, in order to model a wide variety of applications. Also, we allow jobs to have either positive or zero interruption time, and either positive, zero, or negative switching time.

The following example and accompanying figure illustrate our multitasking function.

Example 1: $n = 3$, $(p_1, p_2, p_3) = (2, 4, 10)$, $f(|S_j|) = |S_j|$, $g_1(p'_1) = (0.1)p'_1$, $g_2(p'_2) = (0.1)p'_2$, and $g_3(p'_3) = (0.1)p'_3$. Suppose we process job 1 first, followed by job 2 and then job 3. If there is no multitasking, then the completion times of jobs 1, 2, and 3 are 2, 6, and 16, respectively, as shown in Figure 1(a). If there is multitasking, then job 1 completes at time $p_1 + |\{2, 3\}| + (0.1)p_2 + (0.1)p_3 = 2 + 2 + 0.4 + 1 = 5.4$. Here, p_1 is the processing time of job 1; $|\{2, 3\}|$ is the switching time; $(0.1)p_2$ is the amount of interruption by job 2; and $(0.1)p_3$ is the amount of interruption by job 3. Job 2

completes at time $5.4 + (0.9)p_2 + |\{3\}| + (0.1)(0.9)p_3 = 5.4 + 3.6 + 1 + 0.9 = 10.9$. Here, $(0.9)p_2$ is the remaining processing time of job 2; $|\{3\}|$ is the switching time; and $(0.1)(0.9)p_3$ is the amount of interruption by job 3, which has a remaining processing time of $(0.9)p_3$ before job 2 becomes a primary job. Job 3 completes at time $10.9 + (0.9)(0.9)p_3 = 10.9 + 8.1 = 19$, where $(0.9)(0.9)p_3$ is the remaining processing time of job 3. The schedule is depicted in Figure 1(b).

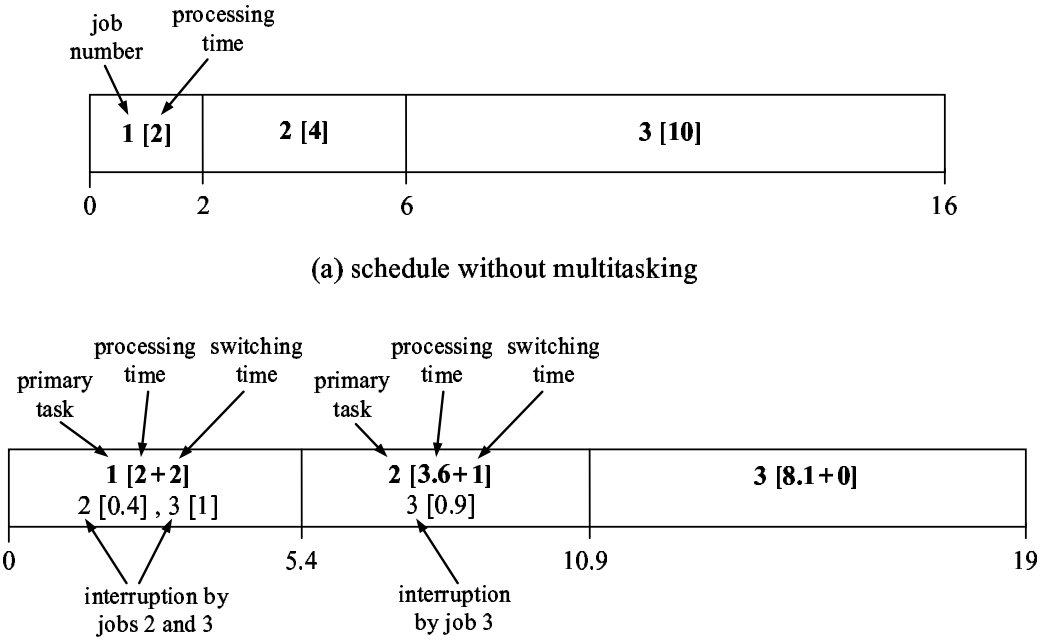

Figure 1: Schedules Without and With Multitasking.

In our model, each job has its own general $g(\cdot)$ function which depends on the characteristic of the job. We now discuss some practical example functions. For some waiting job i, we may have $g_i(p_i') = 0$, which means that job i is not processed at all when another primary job is scheduled. If $g_i(p_i') = 0$ and $f(\cdot) \equiv 0$, then job i never interrupts other jobs. If $g_i(p_i') = 0$ and $f(\cdot) \not\equiv 0$, then job i does delay other jobs while it is waiting for processing, but it only contributes to the switching time. This represents a practical situation where the worker needs to spend time maintaining the waiting job i. This time might be spent, for example, answering inquiries from the owner of job i about the status of the job, or filing extensions to extend the waiting time. However, there is no processing of any portion of job i during the processing of the primary job. For another job i, we

may have $g_i(p'_i) = c$, where c is a positive constant such that $nc < p_i$. In this case, job i always interrupts other jobs by a constant amount while it is waiting for processing, as in Coffman et al. (1970). Another possible interruption function is $g_i(p'_i) = Dp'_i$, where $0 < D < 1$ is a constant, as demonstrated in Example 1. In this case, a fixed proportion of the waiting job is processed during the processing of the primary job. This function enables equitable treatment of the waiting tasks. As confirmation of the practical relevance of our model, we now relate it closely to two studies of interruptions within the operations management literature.

First, Seshadri and Shapira (2001) consider the management of a long-term process, where interruptions arise from the need to maintain several short-term processes. All processes show improvement at a known rate when attended, but decay at a different known rate when unattended. Within the context of our model, we may view the short-term processes as interrupting jobs. Seshadri and Shapira (2001) derive, in Lemma 1 of their paper, a formula for the amount of interruption that is necessary to sustain a given short-term process. This formula is a special case of our function $g_i(\cdot)$.

Second, Dobson et al. (2013) study a service process with three stages: investigation, backroom testing, and further investigation and conclusion. Applications arise in hospital emergency departments and various professional services. Customers within the system, who are analogous to our waiting jobs, interrupt the investigator's work. The authors model the rate of interruption as proportional to the number of customers in the system, which is equivalent to $|S_j|$ in our model. Thus, their model is consistent with both our switching function $f(|S_j|)$ and our interruption function $\sum_{i \in S_j} g_i(p'_i)$.

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 $L_j(\sigma) = C_j(\sigma) - d_j$. We let $U_j(\sigma)$ be a binary variable that indicates whether job j is late or on-time; thus, $U_j(\sigma) = 1$ 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.

When $f(\cdot) \equiv 0$ and $g_i(\cdot) \equiv 0$ for all i, a scheduling problem with multitasking reduces to the corresponding classical scheduling problem. This implies the following result.

Remark 1 *If the recognition version of a classical scheduling problem without multitasking is binary (respectively, unary) NP-complete (Garey and Johnson 1979), then the recognition version of*

the corresponding problem with multitasking is also binary (respectively, unary) NP-complete.

We consider the minimization of four objective functions: the total weighted completion time $\sum_{j=1}^{n} w_j C_j$, the maximum lateness $L_{\max} = \max_{1 \le j \le n}\{L_j\}$, the number of late jobs $\sum_{j=1}^{n} U_j$, and the weighted number of late jobs $\sum_{j=1}^{n} w_j U_j$. All four are among the most widely studied and best practically motivated objectives within the scheduling literature (Brucker 2007, Pinedo 2012). Using the three-field $\alpha \mid \beta \mid \gamma$ notation of Graham et al. (1979), the classical versions of these scheduling problems are denoted by $1 \mid\mid \sum w_j C_j$, $1 \mid\mid L_{\max}$, $1 \mid\mid \sum U_j$, and $1 \mid\mid \sum w_j U_j$, respectively. We use "*mt*" in the β field to denote a multitasking environment.

The following indexing rules from classical scheduling theory are useful in our work. A shortest weighted processing time (SWPT) sequence schedules the jobs such that $w_1/p_1 \ge w_2/p_2 \ge \cdots \ge w_n/p_n$ (Smith 1956). An earliest due date (EDD) sequence schedules the jobs such that $d_1 \le d_2 \le \cdots \le d_n$ (Jackson 1955).

4 Total Weighted Completion Time

In this section, we consider the total weighted completion time objective, i.e., problem $1 \mid mt \mid \sum w_j C_j$. For the classical $1 \mid\mid \sum w_j C_j$ problem, the SWPT rule generates an optimal schedule (Smith 1956). However, in the presence of multitasking, this rule is not generally optimal. For example, consider the following instance, which is an extension of Example 1: $n = 3$, $(p_1, p_2, p_3) = (2, 4, 10)$, $f(|S_j|) = |S_j|$, $g_1(p'_1) = (0.1)p'_1$, $g_2(p'_2) = (0.1)p'_2$, $g_3(p'_3) = (0.1)p'_3$, and $(w_1, w_2, w_3) = (10, 19, 1)$. The SWPT schedule is $(1, 2, 3)$, with a cost of $w_1C_1 + w_2C_2 + w_3C_3 = (10)(5.4) + (19)(10.9) + (1)(19) = 280.1$. However, an optimal schedule is $(2, 1, 3)$, with a cost of $w_2C_2 + w_1C_1 + w_3C_3 = (19)(7.2) + (10)(10.9) + (1)(19) = 264.8$.

We now describe an efficient optimal algorithm for problem $1 \mid mt \mid \sum w_j C_j$. The algorithm schedules the jobs backwards. We use the variable k to indicate the number of jobs that have already been scheduled. After a job is scheduled, k is incremented by one. We let R denote the list of jobs that have been scheduled. When the algorithm terminates, R gives the final sequence of jobs. At each iteration, among all the unscheduled jobs, we select a job i with the smallest $w_i/[h_i(n-k-1) + f(k) + \sum_{y \in R} g_y(h_y(n-k-1))]$ value. In the denominator of this formula,

the first term represents the remaining processing time of job i. The second term represents the switching time during the processing of job i, which is a function of the number of interrupting jobs, $k = |R|$. The third term represents the total amount of interruption by the jobs of R, which depends on their remaining processing times. We let J denote the set of jobs that have not yet been scheduled. To reduce the running time of the algorithm, we first compute $h_i(j)$, which stores the remaining processing time of job i after it has interrupted j primary jobs, for $1 \leq i \leq n$ and $0 \leq j \leq n-1$. The details of the algorithm are given below, where the symbol "||" denotes the concatenation operator that appends one list to another.

Algorithm WC

1. $J := \{1, 2, \ldots, n\}$, $R := ()$, $k := 0$.
2. For $i \in J$ do
   (a) $h_i(0) := p_i$.
   (b) For j from 1 to $n-1$ do
   $$h_i(j) := h_i(j-1) - g_i(h_i(j-1)).$$
3. $z := 0$.
4. For $y \in R$ do
   $$z := z + g_y(h_y(n-k-1)).$$
5. $i := \arg\min_{l \in J} \{\frac{w_l}{h_l(n-k-1)+f(k)+z}\}$, with ties broken arbitrarily.
6. $J := J \setminus \{i\}$, $R := (i)||R$, $k := k+1$.
7. If $|R| < n$, then go to Step 3.
8. Schedule the jobs, with multitasking, in the order in which they appear in R.

Theorem 1 *Algorithm WC finds an optimal schedule for problem* $1 \mid mt \mid \sum w_j C_j$ *in* $O(n^2)$ *time, if the* $f(\cdot)$ *and* $g_j(\cdot)$ *functions can each be computed in constant time.*

Proof. It suffices to consider only schedules with no idle time preceding any job. By contradiction, suppose there exists an optimal schedule σ in which there is a pair of adjacent jobs i and j such that i is processed before j, and

$$\frac{w_i}{h_i(n-k-1) + f(k) + \sum_{y \in R} g_y(h_y(n-k-1))} < \frac{w_j}{h_j(n-k-1) + f(k) + \sum_{y \in R} g_y(h_y(n-k-1))},$$

where R is the set of jobs scheduled after j, and $k = |R|$. We interchange jobs i and j to obtain another schedule σ'. Let t denote the start time of job i in schedule σ. Note that there are $n-k-2$ jobs processed before job i. We have

$$C_i(\sigma) = t + h_i(n-k-2) + f(k+1) + g_j(h_j(n-k-2)) + \sum_{y \in R} g_y(h_y(n-k-2)).$$

On the right hand side of the above equation, the second term is the remaining processing time of job i, and the third term is the switching time of the $k+1$ jobs in $R \cup \{j\}$. The fourth and the fifth terms are the interruption due to job j and the interruption due to the k jobs scheduled after job j, respectively. Similarly, we have

$$\begin{aligned} C_j(\sigma) \;=\; & t + h_i(n-k-2) + f(k+1) + g_j(h_j(n-k-2)) + \sum_{y \in R} g_y(h_y(n-k-2)) \\ & + h_j(n-k-1) + f(k) + \sum_{y \in R} g_y(h_y(n-k-1)), \end{aligned}$$

$$C_j(\sigma') = t + h_j(n-k-2) + f(k+1) + g_i(h_i(n-k-2)) + \sum_{y \in R} g_y(h_y(n-k-2)),$$

and

$$\begin{aligned} C_i(\sigma') \;=\; & t + h_j(n-k-2) + f(k+1) + g_i(h_i(n-k-2)) + \sum_{y \in R} g_y(h_y(n-k-2)) \\ & + h_i(n-k-1) + f(k) + \sum_{y \in R} g_y(h_y(n-k-1)). \end{aligned}$$

Observing that $h_y(l) = h_y(l-1) - g_y(h_y(l-1))$, for $1 \le y \le n$ and $0 < l < n$, we have $C_j(\sigma) = C_i(\sigma')$. Moreover, after processing jobs i and j, the remaining processing times of each job in R are identical in schedules σ and σ'. Hence, the completion time of any job, other than jobs i and j, does not change between the two schedules. Thus, we have

$$\begin{aligned} \textstyle\sum_{l=1}^{n} w_l C_l(\sigma) - \sum_{l=1}^{n} w_l C_l(\sigma') \;&=\; [w_i C_i(\sigma) + w_j C_j(\sigma)] - [w_i C_i(\sigma') + w_j C_j(\sigma')] \\ &=\; w_j[h_i(n-k-1) + f(k) + \textstyle\sum_{y \in R} g_y(h_y(n-k-1))] \\ &\quad - w_i[h_j(n-k-1) + f(k) + \textstyle\sum_{y \in R} g_y(h_y(n-k-1))] \\ &>\; 0. \end{aligned}$$

Hence, $\sum_{l=1}^{n} w_l C_l(\sigma) > \sum_{l=1}^{n} w_l C_l(\sigma')$, contradicting the assumption that σ is an optimal schedule. Therefore, Algorithm WC finds an optimal schedule. Our job interchange argument generalizes

that of Smith (1956), and works because the total switching time is unchanged as a result of the interchange.

Step 1 of the algorithm requires constant time. Step 2 requires $O(n^2)$ time. Steps 3–7 are repeated n times. Step 3 requires constant time. Steps 4 and 5 each require $O(n)$ time. Steps 6 and 7 require constant time. Thus, Steps 3–7 require a total of $O(n^2)$ time. Since each of the n primary jobs can be interrupted up to $n-1$ times, Step 8 requires $O(n^2)$ time. Hence, the overall running time of the algorithm is $O(n^2)$. □

5 Maximum Lateness

In this section, we consider the maximum lateness objective, i.e., problem $1 \mid mt \mid L_{\max}$. Observe that both the on-time jobs and the late jobs are eventually processed, and before they become primary jobs these two types of jobs are indistinguishable from each other. Hence, we assume that both on-time and late jobs can interrupt other jobs. For the classical $1 \mid\mid L_{\max}$ problem, the EDD rule generates an optimal schedule (Jackson 1955). The following result states that this scheduling rule remains optimal in the presence of multitasking.

Theorem 2 *For problem $1 \mid mt \mid L_{\max}$, there exists an optimal schedule that uses the EDD rule, with no inserted idle time.*

Proof. Since $L_j(\sigma)$ is a nondecreasing function of $C_j(\sigma)$ for all j, there exists an optimal schedule with no inserted idle time. Suppose, to the contrary, that there exists an optimal schedule σ that does not follow the EDD rule. Then, there must exist a pair of adjacent jobs i and j such that i is processed before j and $d_i > d_j$. We interchange job i with job j to obtain another schedule σ'. Let t denote the start time of job i in schedule σ. Let R denote the set of jobs scheduled after job j in σ, and let $k = |R|$. The calculation of the job completion times below is similar to that in the proof of Theorem 1. We have

$$C_i(\sigma) = t + h_i(n-k-2) + f(k+1) + g_j(h_j(n-k-2)) + \sum_{y \in R} g_y(h_y(n-k-2)),$$

$$\begin{aligned} C_j(\sigma) \;=\; & t + h_i(n-k-2) + f(k+1) + g_j(h_j(n-k-2)) + \sum_{y \in R} g_y(h_y(n-k-2)) \\ & + h_j(n-k-1) + f(k) + \sum_{y \in R} g_y(h_y(n-k-1)), \end{aligned}$$

$$C_j(\sigma') = t + h_j(n-k-2) + f(k+1) + g_i(h_i(n-k-2)) + \sum_{y \in R} g_y(h_y(n-k-2)),$$

and

$$\begin{aligned} C_i(\sigma') \;=\; & t + h_j(n-k-2) + f(k+1) + g_i(h_i(n-k-2)) + \sum_{y \in R} g_y(h_y(n-k-2)) \\ & + h_i(n-k-1) + f(k) + \sum_{y \in R} g_y(h_y(n-k-1)). \end{aligned}$$

Observing that $h_y(l) = h_y(l-1) - g_y(h_y(l-1))$ for each $1 \leq y \leq n$ and $0 < l < n$, we have $C_j(\sigma) = C_i(\sigma')$. Moreover, after processing jobs i and j, the remaining processing times of each job in R are identical in both schedules. Hence, the completion time of any job, other than jobs i and j, does not change between the two schedules. Thus,

$$L_j(\sigma) - L_j(\sigma') = [C_j(\sigma) - d_j] - [C_j(\sigma') - d_j] = C_j(\sigma) - C_j(\sigma') = C_i(\sigma') - C_j(\sigma') \geq 0.$$

In addition,

$$L_j(\sigma) - L_i(\sigma') = [C_j(\sigma) - d_j] - [C_i(\sigma') - d_i] = d_i - d_j > 0.$$

Hence, $\max\{L_i(\sigma), L_j(\sigma)\} \geq \max\{L_i(\sigma'), L_j(\sigma')\}$, which implies $L_{\max}(\sigma') \leq L_{\max}(\sigma)$. Repeating this interchange, we transform σ into an EDD schedule σ' with optimal cost, which contradicts our assumption that an EDD schedule is not optimal. Our job interchange argument generalizes that of Jackson (1955), and works because the total switching time is unchanged as a result of the interchange. □

Theorem 2 implies that an optimal sequence for problem $1 \mid mt \mid L_{\max}$ can be found in $O(n \log n)$ time. However, constructing an optimal schedule, including the start times of the primary jobs, requires $O(n^2)$ time, if the $f(\cdot)$ and $g_j(\cdot)$ functions can be computed in constant time.

6 Number of Late Jobs

In this section, we consider the objective of minimizing the number of late jobs, i.e., problem $1 \mid mt \mid \sum U_j$. We assume that late jobs are processed at the end of the schedule, instead of being discarded. As in Section 5, we assume that late jobs can interrupt other jobs. We first state a property of an optimal schedule.

Lemma 1 *For problem $1 \mid mt \mid \sum U_j$, there exists an optimal schedule in which all on-time jobs are scheduled in EDD order, followed by all late jobs in arbitrary order, with no inserted idle time.*

Proof. Since $U_j(\sigma)$ is a nondecreasing function of $C_j(\sigma)$ for all j, there exists an optimal schedule with no inserted idle time. For a given instance of the problem, let E denote the set of on-time jobs and T the set of late jobs. A pairwise interchange argument shows that there exists an optimal schedule where all the jobs of E precede all the jobs of T. Thus, the total amount of interruption of the jobs of E by the jobs of T is independent of the sequence of the jobs of T. Consequently, the number of jobs of E is also independent of the sequence of the jobs of T. Hence, without loss of generality, the jobs of T can be 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 in such a way that their maximum lateness is minimized. Therefore, from Theorem 2, there exists an optimal schedule where the jobs in E are scheduled by the EDD rule. □

The classical $1 \mid\mid \sum U_j$ problem can be solved in $O(n \log n)$ time using the Hodgson-Moore algorithm (Moore 1968). However, in the presence of multitasking, the problem is *NP*-hard.

Theorem 3 *The recognition version of* $1 \mid mt \mid \sum U_j$ *is binary* NP*-complete.*

Proof. See the online Appendix.

Although problem $1 \mid mt \mid \sum U_j$ is *NP*-hard in general, it is solvable in polynomial time for some important special cases, for example the case where $g_i(p'_i) = c$ for all i, and the case where $g_i(p'_i) = Dp'_i$ for all i. We now present an $O(n \log n)$ time algorithm, Algorithm U, for the case where $g_i(p'_i) = Dp'_i$ for $i \in N$ and $0 < D < 1$, while $f(|S_j|)$ can be positive, zero, or negative for any nonempty $S_j \subset N$.

Consider the interruption function $g_i(p'_i) = Dp'_i$. Let job J_i be a waiting job. The first time it interrupts a primary job, the amount of interruption is Dp_i. After the interruption, the remaining processing time of job J_i is $p_i - Dp_i = (1 - D)p_i$. The second time it interrupts a primary job, the amount of interruption is $D(1 - D)p_i$. After the second interruption, the remaining processing time of job J_i is $(1 - D)p_i - D(1 - D)p_i = (1 - D)^2 p_i$. Similarly, the kth time it interrupts a primary job, the amount of interruption is $D(1 - D)^{k-1} p_i$, and the remaining processing time after the interruption is $(1 - D)^k p_i$.

Algorithm U described below is a generalization of the Hodgson-Moore algorithm. We let L denote the list of unscheduled jobs sorted in EDD order and, in case of a tie, in SPT order. Jobs

are deleted from the front of L, one at a time. We use E and T to denote the sets of on-time and late jobs, respectively. Initially, $E = T = \emptyset$. We use $\hat{P}$ to denote the total processing time of the jobs in L, and P' to denote the total processing time of the jobs in T. Initially, $\hat{P} = P$ and $P' = 0$. Finally, we let k denote the number of on-time jobs scheduled so far.

The algorithm considers each job in L in turn. Suppose the last job completed is job j. If job j meets its due date, then we consider the next job $j+1$. Otherwise, we delete the longest completed job, which may or may not be job j, from the schedule, and then consider the next job $j+1$.

Algorithm U

1. $L := N$. Sort the jobs of L in EDD order, with ties broken by SPT order.
2. $E := \emptyset$, $T := \emptyset$, $\hat{P} := \sum_{j \in L} p_j$, $P' := 0$, $k := 0$, $t := 0$.
3. While $L \neq \emptyset$ do
   (a) Let j be the first job in L. Remove j from L. $\hat{P} := \hat{P} - p_j$. Add j to E.
   (b) $t := t + f(n-k-1) + (1-D)^k p_j + D(1-D)^k(\hat{P} + P')$.
   (c) If $t \leq d_j$, then $k := k + 1$.
   (d) Otherwise, let q be the longest job in E; remove job q from E and add it to T; $t := t - f(n-k-1) - (1-D)^k p_q - D(1-D)^k(\hat{P} + P')$; $P' := P' + p_q$.
4. Schedule the jobs of E, in EDD order, with multitasking. Then, schedule the jobs of T, in arbitrary order, with multitasking.

For any job subset $S \subseteq N$, let $\ell(S)$ denote the completion time of the last job in S if we schedule the jobs in S in front of the jobs in $N \setminus S$ according to Algorithm U with no idle time between the jobs. In order to prove the optimality of Algorithm U, we need the following preliminary result.

Lemma 2 *For an instance of problem* $1 \mid mt \mid \sum U_j$ *where* $g_i(p'_i) = Dp'_i$ *for* $i \in N$, *consider the job subset* S *and any two jobs* k, k' *in* S. *If* $p_k \leq p_{k'}$, *then* $\ell(S \setminus \{k\}) \geq \ell(S \setminus \{k'\})$.

Proof. Let $T = N \setminus S$. Suppose job k is the rth job in S and is removed from S. The resulting effect on $\ell(S)$ is as follows:

(i) The total switching time during the processing of the first $r-1$ jobs in S is reduced by $f(n-r)$.

(ii) The amount of processing time $(1-D)^{r-1}p_k$ of primary job k is excluded from $\ell(S)$.

(iii) Since each waiting job $i \in T$ interrupts one less job in S and therefore the amount of interruption by job i is reduced by $D(1-D)^{|S|-1}p_i$, the total amount of interruption caused by the waiting jobs in T is reduced by $\sum_{i\in T} D(1-D)^{|S|-1}p_i$.

Further, when job k is removed from S, it is added to T, and hence interrupts the last $|S|-r$ jobs in S. The resulting effect on $\ell(S)$ is as follows:

(iv) The total switching time during the processing of the last $|S|-r$ jobs in S is increased by $\sum_{j=r}^{|S|-1}[f(n-j)-f(n-j-1)]$.

(v) The total amount of interruption by the last $|S|-r$ jobs in S is increased by $\sum_{j=r}^{|S|-1} D(1-D)^{j-1}p_k$.

Summarizing (i)–(v), we have

$$\begin{aligned}\ell(S\setminus\{k\}) &= \ell(S) - f(n-r) - (1-D)^{r-1}p_k - \textstyle\sum_{i\in T} D(1-D)^{|S|-1}p_i \\ &\quad + \textstyle\sum_{j=r}^{|S|-1}[f(n-j)-f(n-j-1)] + \sum_{j=r}^{|S|-1} D(1-D)^{j-1}p_k \\ &= \ell(S) - f(n-|S|) - (1-D)^{|S|-1}p_k - \textstyle\sum_{i\in T} D(1-D)^{|S|-1}p_i,\end{aligned}$$

which is independent of r. Similarly,

$$\ell(S\setminus\{k'\}) = \ell(S) - f(n-|S|) - (1-D)^{|S|-1}p_{k'} - \sum_{i\in T} D(1-D)^{|S|-1}p_i.$$

Since $p_k \le p_{k'}$, we have $\ell(S\setminus\{k\}) \ge \ell(S\setminus\{k'\})$. □

We are now ready to prove the main result of this section.

Theorem 4 *Algorithm U finds an optimal schedule for problem* $1 \mid mt \mid \sum U_j$ *where* $g_i(p_i') = Dp_i'$ *for* $i \in N$*, in* $O(n \log n)$ *time.*

Proof. We first prove the validity of Algorithm U. The proof is an extension of a proof of optimality of Moore's algorithm (Pinedo 2012, pp. 48–49). It works because Lemma 2 shows that changes to the schedule length are independent of the position of the discarded job.

Let the jobs be indexed in EDD order. From Lemma 1, we only consider on-time jobs in index order. A job subset S is said to be feasible if all the jobs in S meet their due dates when they are scheduled before the jobs in $N\setminus S$ and are sequenced in index order. For any $l \le n$, set S is said

to be *l-optimal* if it is a feasible subset of jobs $1, 2, \ldots, l$ and if it has, among all feasible subsets of jobs $1, 2, \ldots, l$, the maximum number of jobs.

The proof consists of three parts. The first part shows by induction that the job set E created in Step 3 at each iteration of Algorithm U is feasible. Let E_j be the set E created in the jth iteration of Algorithm U. Clearly, E_1 is feasible. For the induction hypothesis, we assume that E_{j-1} is feasible, where $1 < j \leq n$. We consider the jth iteration of Algorithm U, where job j is added to E. In Step 3(c) of this iteration, if $t \leq d_j$, then clearly E_j is feasible. Alternatively, suppose $t > d_j$. From the induction hypothesis, all the jobs in $E \setminus \{j\}$ must meet their due dates after the longest job q is removed from E. Because $p_j \leq p_q$, by Lemma 2, the completion time of the last job in E after Step 3(d) is no greater than the completion time of the last job in E before Step 3(a). Furthermore, d_j is no less than d_{j-1}, which is no less than the completion time of the last job in E before Step 3(a). Thus, job j also meets its due date. Hence, E_j is feasible.

The second part of the proof shows by induction that for any j and l such that $0 \leq j < l \leq n$, there exists an l-optimal set which is a subset of $E_j \cup \{j+1, j+2, \ldots, l\}$, where $E_0 = \emptyset$. Clearly, this is true when $j = 0$. For the induction hypothesis, we assume that there exists an l-optimal set J' such that $J' \subseteq E_{j-1} \cup \{j, j+1, \ldots, l\}$, where $0 < j < l$. We show that an l-optimal set J'' can be created from the jobs in E_j and some of jobs $j+1, j+2, \ldots, l$ by considering three cases:

Case 1: Set E_j is equal to set E_{j-1} plus job j. In this case, we set $J'' = J'$.

Case 2: Set E_j is equal to set E_{j-1}, plus job j, minus some job q which is not an element of J'. Again, we set $J'' = J'$.

Case 3: Set E_j is equal to set E_{j-1} plus job j minus some job q which is an element of J'. Since $E_{j-1} \cup \{j\}$ is not a feasible set, there exists a job $r \in E_{j-1} \cup \{j\}$ such that $r \notin J'$. We let $J'' = (J' \cup \{r\}) \setminus \{q\}$. Clearly, J'' is a subset of $E_j \cup \{j+1, j+2, \ldots, l\}$. Furthermore, the number of jobs in J'' is the same as the number of jobs in J'. Thus, to show that J'' is l-optimal, we need only show that J'' is feasible. Let

$$\hat{J} = J' \cap \{j+1, j+2, \ldots, l\} = J'' \cap \{j+1, j+2, \ldots, l\}.$$

It suffices to show the following two properties: (i) The set $J'' \cap \{1, 2, \ldots, j\}$ is feasible. (ii) The completion time of each of the jobs in $\hat{J}$, when the jobs in $\hat{J}$ are scheduled after the jobs in $J'' \cap \{1, 2, \ldots, j\}$, is no greater than its completion time when the jobs in $\hat{J}$ are scheduled after the

jobs in $J' \cap \{1, 2, \ldots, j\}$. Property (i) holds because $J'' \cap \{1, 2, \ldots, j\}$ is a subset of E_j, and from the first part of the proof the set E_j is feasible. Property (ii) follows from Lemma 2, the fact that J'' differs from set J' only in elements r and q, and the fact that $p_r \leq p_q$.

The third part of the proof shows by induction that set E_j is j-optimal for $j = 1, 2, \ldots, n$. Clearly, E_j is j-optimal for $j = 0$ and $j = 1$. Suppose E_{j-1} is $(j-1)$-optimal. From the second part of the proof, the set $E_{j-1} \cup \{j\}$ must contain a j-optimal set. If $E_{j-1} \cup \{j\}$ is feasible, then $E_j = E_{j-1} \cup \{j\}$ and is j-optimal, since E_{j-1} is $(j-1)$-optimal, and at most one more job can be on time in E_j relative to E_{j-1}. If $E_{j-1} \cup \{j\}$ is not feasible, then the j-optimal set must be a proper subset of $E_{j-1} \cup \{j\}$ and must contain as least as many jobs as E_{j-1}. Clearly, E_j satisfies this condition and is, therefore, j-optimal.

We now consider the running time of Algorithm U. Step 1 requires $O(n \log n)$ time. Step 2 requires $O(n)$ time. The while loop in Step 3 is executed n times. Inside the while loop, the most time-consuming step is finding a 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. Step 4 requires $O(n)$ time. Therefore, the overall running time of Algorithm U is $O(n \log n)$. □

Remark 2 *With a simple modification to the completion time update in Steps 3(b) and 3(d), Algorithm U solves problem* $1 \mid mt \mid \sum U_j$ *for the special case where for all* i, $g_i(p'_i) = c$, *and* c *is a constant independent of* p'_i.

7 Weighted Number of Late Jobs

We now consider the objective of minimizing the weighted number of late jobs, i.e., problem $1 \mid mt \mid \sum w_j U_j$. As in Section 6, we assume that late jobs are processed at the end of the schedule and can interrupt other jobs. It is easy to verify that Lemma 1 remains valid for this problem.

The classical $1 \mid\mid \sum w_j U_j$ problem is known to be binary *NP*-hard (Karp 1972), but is solvable in $O(n \min\{W, P\})$ time, where $W = \sum_{j=1}^{n} w_j$ and $P = \sum_{j=1}^{n} p_j$ (Lawler and Moore 1969). However, in the presence of multitasking, the problem is unary *NP*-hard.

Theorem 5 *The recognition version of* $1 \mid mt \mid \sum w_j U_j$ *is unary* NP*-complete.*

Proof. See the online Appendix.

Although $1 \mid mt \mid \sum w_j U_j$ is unary *NP*-hard in general, the problem is solvable in pseudo-polynomial time for some important special cases, for example the case where $g_i(p'_i) = c$ for all i, and the case where $g_i(p'_i) = Dp'_i$ for all i. We now present two pseudo-polynomial time algorithms, WU1 and WU2, for problem $1 \mid mt \mid \sum w_j U_j$ where $g_i(p'_i) = Dp'_i$ for all i. Which of the two algorithms is more efficient depends on the problem parameters. Note that the problem is equivalent to maximizing the weighted number of on-time jobs. Consequently, the first dynamic programming algorithm has a maximization objective.

Algorithm WU1

Preprocessing

Sequence and index the jobs in EDD order.

Optimal Value Function

Define $e(j,k,t)$ as the maximum possible total weight of on-time jobs in a feasible schedule that (i) has considered jobs $1, 2, \ldots, j$, (ii) has scheduled k jobs on time, and (iii) has a total processing time of scheduled on-time jobs equal to t, for $j, k = 0, 1, \ldots, n$ and $t = 0, 1, \ldots, P$, where $e(j,k,t) = -\infty$ if no such schedule exists.

Recurrence Relation

$$e(j,k,t) = \begin{cases} \max\{w_j + e(j-1,k-1,t-p_j),\ e(j-1,k,t)\}, \\ \qquad \text{if } k > 0,\ p_j \le t \le P, \text{ and } t + \sum_{i=1}^{k} f(n-i) + (P-t)[1-(1-D)^k] \le d_j; \\ e(j-1,k,t), \quad \text{otherwise.} \end{cases}$$

Boundary Conditions

$$e(0,0,t) = \begin{cases} 0, & \text{if } t = 0; \\ -\infty, & \text{if } t > 0; \end{cases}$$

$$e(j,k,t) = -\infty \text{ if } j < k.$$

Optimal Solution Value

$$\max_{k=0,1,\ldots,n;\ t=0,1,\ldots,P} \{e(n,k,t)\}.$$

Lemma 3 *If $g_i(p'_i) = Dp'_i$ for all $i \in N$, then Algorithm WU1 finds an optimal schedule for problem $1 \mid mt \mid \sum w_j U_j$ in $O(n^2 P)$ time.*

Proof. Recall that Lemma 1 is also valid for problem $1 \mid mt \mid \sum w_j U_j$. Hence, it suffices to consider schedules in which the on-time jobs are sequenced in EDD order with no inserted machine idle time. Consider the subproblem containing only jobs $1, 2, \ldots, j$, where k out of these j jobs are required to be on-time, and the total processing time of these k on-time jobs is required to be exactly t. We can either choose job j to be on-time and processed immediately after the other $k-1$ on-time job, or choose job j to be late. If we choose job j to be on-time, then the total amount of interruption that job j and the other $k-1$ scheduled on-time jobs encounter is $(P-t)[1-(1-D)^k]$, and the total amount of switching time that these k on-time jobs encounter is $\sum_{i=1}^{k} f(n-i)$. Hence, the completion time of job j is $t + \sum_{i=1}^{k} f(n-i) + (P-t)[1-(1-D)^k]$. Therefore, if $k > 0$, $t \geq p_j$, and $t + \sum_{i=1}^{k} f(n-i) + (P-t)[1-(1-D)^k] \leq d_j$, then we can either choose job j to be on-time, in which case the total weight of on-time jobs becomes $w_j + e(j-1, k-1, t-p_j)$; or, choose job j to be late, in which case the total weight of on-time jobs remains $e(j-1, k, t)$. Otherwise, we can only assign job j to be a late job. Thus, $e(j,k,t) = \max\{w_j + e(j-1, k-1, t-p_j),\ e(j-1,k,t)\}$ in the first scenario and $e(j,k,t) = e(j-1,k,t)$ in the second scenario, as in the recurrence relation. Hence, Algorithm WU1 compares the value of all possible state transitions with the jobs in EDD order, and therefore finds an optimal schedule for problem $1 \mid mt \mid \sum w_j U_j$.

To consider the running time of Algorithm WU1, we note that $j, k \leq n$ and $t \leq P$. Since the recurrence relation requires only constant time, the running time of Algorithm WU1 is $O(n^2 P)$. □

Algorithm WU2

Preprocessing

Sequence and index the jobs in EDD order.

Optimal Value Function

Define $e(j,k,u)$ as the minimum possible total duration of the on-time jobs in a feasible schedule that (i) has considered jobs $1, 2, \ldots, j$, (ii) has scheduled k jobs on time, and (iii) has a total weight of scheduled on-time jobs no less than u, for $j, k = 0, 1, \ldots, n$ and $u = 0, 1, \ldots, W$, where $e(j,k,u) = +\infty$ if no such schedule exists.

Recurrence Relation

$$e(j,k,u) = \begin{cases} \min\{(1-D)e(j-1,k-1,u-w_j)+f(n-k)+D\sum_{i=1}^{k-1} f(n-i)+(1-D)^k p_j+DP, \\ \qquad e(j-1,k,u)\}, \quad \text{if } k>0,\ u \geq w_j, \text{ and } (1-D)e(j-1,k-1,u-w_j) \\ \qquad\qquad + f(n-k) + D\sum_{i=1}^{k-1} f(n-i) + (1-D)^k p_j + DP \leq d_j; \\ e(j-1,k,u), \quad \text{otherwise.} \end{cases}$$

Boundary Conditions

$$e(0,0,u) = \begin{cases} 0, & \text{if } u = 0; \\ +\infty, & \text{if } u > 0; \end{cases}$$

$$e(j,k,u) = +\infty \text{ if } j < k.$$

Optimal Solution Value

$$\max\{u \mid e(n,k,u) < +\infty, \text{ for some } k = 0,1,\ldots,n\}.$$

Lemma 4 *If $g_i(p_i') = Dp_i'$ for all $i \in N$, then Algorithm WU2 finds an optimal schedule for problem $1 \mid mt \mid \sum w_j U_j$ in $O(n^2W)$ time.*

Proof. As in the proof of Lemma 3, it suffices to consider schedules in which the on-time jobs are sequenced in EDD order with no inserted machine idle time. Consider the subproblem containing only jobs $1,2,\ldots,j$, where k out of these j jobs are required to be on-time, and the total weight of these k on-time jobs is required to be at least u. We can either choose job j to be on-time and processed immediately after the other $k-1$ on-time jobs, or choose job j to be late. If job j is on time, the total duration of the on-time jobs is $(1-D)^{k-1}p_j + f(n-k) + D[P - e(j-1,k-1,u-w_j) + \sum_{i=1}^{k-1} f(n-i) - (1-D)^{k-1}p_j]$, where the first term is the remaining processing time of job j after it has interrupted $k-1$ other jobs, the second term is the switching time, and the third term is the interruption by the remaining jobs during the processing of job j. In this case, the completion time of job j is

$$\begin{aligned} & e(j-1,k-1,u-w_j) + (1-D)^{k-1}p_j + f(n-k) \\ & + D[P - e(j-1,k-1,u-w_j) + \textstyle\sum_{i=1}^{k-1} f(n-i) - (1-D)^{k-1}p_j] \\ & = (1-D)e(j-1,k-1,u-w_j) + f(n-k) + D\textstyle\sum_{i=1}^{k-1} f(n-i) + (1-D)^k p_j + DP. \end{aligned}$$

Thus, if $k>0$, $u \geq w_j$, and $(1-D)e(j-1,k-1,u-w_j)+f(n-k)+D\sum_{i=1}^{k-1} f(n-i)+(1-D)^k p_j + DP \leq d_j$, then we can either choose job j to be on-time, in which case the total duration of the on-time jobs becomes $(1-D)e(j-1,k-1,u-w_j)+f(n-k)+D\sum_{i=1}^{k-1} f(n-i)+(1-D)^k p_j+DP$;

or, choose job j to be late, in which case the total duration of the on-time jobs remains $e(j-1,k,u)$. Otherwise, we can only choose job j to be late. Thus, $e(j,k,u) = \min\{(1-D)e(j-1,k-1,u-w_j) + f(n-k) + D\sum_{i=1}^{k-1} f(n-i) + (1-D)^k p_j + DP,\ e(j-1,k,u)\}$ in the first scenario, and $e(j,k,u) = e(j-1,k,u)$ in the second scenario, as in the recurrence relation. Hence, Algorithm WU2 compares the cost of all possible state transitions with the jobs in EDD order, and therefore finds an optimal schedule for problem $1 \mid mt \mid \sum w_j U_j$.

To consider the running time of Algorithm WU2, we note that the values of $\sum_{i=1}^{k-1} f(n-i)$, $k = 1, 2, \ldots, n$, can be predetermined in $O(n)$ time. Note also that $j, k \leq n$ and $u \leq W$. Since the recurrence relation requires only constant time, the running time of Algorithm WU2 is $O(n^2 W)$. □

Theorem 6 *If $g_i(p'_i) = Dp'_i$ for all $i \in N$, an optimal schedule for problem $1 \mid mt \mid \sum w_j U_j$ can be found in $O(n^2 \min\{W, P\})$ time.*

Proof. The theorem follows directly from Lemmas 3 and 4. Our proof of optimality generalizes that of Lawler and Moore (1969), and works because the total switching time is independent of the choice of which subset of jobs with a given cardinality is on time. □

Remark 3 *With a simple modification to the feasibility condition for the first choice in the recurrence relation, Algorithms WU1 and WU2 solve problem $1 \mid mt \mid \sum w_j U_j$ for the special case where for all i, $g_i(p'_i) = c$, and c is a constant independent of p'_i.*

8 Cost and Value of Multitasking

In this section, we computationally evaluate the amount by which multitasking increases scheduling cost or value. This information informs a company about how much it is worth to invest in systems or processes to reduce multitasking. We consider the total weighted completion time problem, the maximum lateness problem, and the weighted number of late jobs problem in Sections 8.1, 8.2, and 8.3, respectively.

8.1 Total weighted completion time

We consider the total weighted completion time problem, $1 \mid mt \mid \sum w_j C_j$. To evaluate the average cost and value of multitasking under this problem, we perform two computational experiments.

The first experiment does not consider any value of interruption, and we set $f(k) = 0.1 \times k$. The second experiment considers the case where the value of interruption exceeds the shutdown and startup time of a job, hence we set $f(k) = -0.1 \times k$ to reflect a positive net value for each interruption. The remaining parameters are identical in the two experiments. Job processing times are randomly generated, with each p_j being uniformly distributed in $\{50, 51, \ldots, 200\}$, and each w_j being uniformly distributed in $\{1, 2, \ldots, 10\}$. We set $n \in \{10, 20, 40, 80\}$. We introduce a variety of different interruption functions, as for different customers, and study their combined effect. For each value of n, we generate 10% of the jobs with $g_i(p'_i) = 0$, 30% of the jobs with $g_i(p'_i) = c$, 30% of the jobs with $g_i(p'_i) = D\sqrt{p'_i}$, and 30% of the jobs with $g_i(p'_i) = Dp'_i$. To see how the different values of c and D affect the cost or value of multitasking, we let $c \in \{0.1, 0.2, 0.3, 0.4\}$ and $D \in \{0.005, 0.01, 0.015, 0.02\}$. We use the triple $(0, c, D)$ to represent one choice. Thus, we have four choices: $(0, 0.1, 0.005)$, $(0, 0.2, 0.01)$, $(0, 0.3, 0.015)$, and $(0, 0.4, 0.02)$. We generate 300 instances for each combination of n and $(0, c, D)$, for a total of $300 \times 4 \times 4 = 4{,}800$ instances for each experiment.

Let C'_j and C_j denote the optimal completion time of job j in the multitasking problem $1 \mid mt \mid \sum w_j C_j$ and the corresponding classical problem $1 \mid\mid \sum w_j C_j$, respectively. Let $avg((\sum w_j C'_j - \sum w_j C_j)/\sum w_j C_j)$ denote the average value of $(\sum w_j C'_j - \sum w_j C_j)/\sum w_j C_j$ over the 300 test instances for each $(n, 0, c, D)$ quadruple. The quantity $avg((\sum w_j C'_j - \sum w_j C_j)/\sum w_j C_j) \times 100\%$ estimates the expected percentage cost of multitasking, or value of multitasking if the quantity is negative. To obtain the optimal total cost $\sum w_j C'_j$ of the multitasking problem $1 \mid mt \mid \sum w_j C_j$, we run Algorithm WC. To obtain the optimal total cost $\sum w_j C_j$ of the classical model, we apply the SWPT rule. For the unweighted problem $1 \mid mt \mid \sum C_j$, we repeat the above computational study with $w_j = 1$, for $j = 1, 2, \ldots, n$.

The results in Table 1 for the weighted problem show that, when there is no value of interruption, the average cost of multitasking ranges from 1.6% to 4.3% for 10 jobs, and from 13.3% to 29.9% for 80 jobs. The cost of multitasking increases approximately in proportion to the number of jobs. When there is a value of interruption, the net cost for 80 jobs ranges from 2.5% to 19.1%. The proportionate reduction in cost due to considering value is much larger for small values of the c and D parameters. For example, if $n = 80$, the cost reduces from 13.3% to 2.5% for $(0, c, D) = (0, 0.1, 0.005)$, compared to only from 29.9% to 19.1% for $(0, c, D) = (0, 0.4, 0.02)$. The percentage

		$avg((\sum w_jC'_j - \sum w_jC_j)/\sum w_jC_j) \times 100\%$			
		$(0, c, D) =$ $(0, 0.1, 0.005)$	$(0, c, D) =$ $(0, 0.2, 0.01)$	$(0, c, D) =$ $(0, 0.3, 0.015)$	$(0, c, D) =$ $(0, 0.4, 0.02)$
Weighted problem; $f(k) = 0.1 \times k$	$n = 10$	1.6%	2.5%	3.4%	4.3%
	$n = 20$	3.4%	5.4%	7.2%	9.0%
	$n = 40$	6.8%	10.6%	14.0%	17.0%
	$n = 80$	13.3%	19.7%	25.2%	29.9%
Weighted problem; $f(k) = -0.1 \times k$	$n = 10$	0.3%	1.3%	2.2%	3.1%
	$n = 20$	0.8%	2.8%	4.6%	6.4%
	$n = 40$	1.5%	5.2%	8.6%	11.7%
	$n = 80$	2.5%	9.0%	14.4%	19.1%
Unweighted problem; $f(k) = 0.1 \times k$	$n = 10$	1.4%	2.2%	2.9%	3.6%
	$n = 20$	3.0%	4.5%	5.9%	7.2%
	$n = 40$	5.9%	8.6%	10.8%	12.9%
	$n = 80$	11.3%	15.6%	19.2%	22.2%
Unweighted problem; $f(k) = -0.1 \times k$	$n = 10$	0.2%	1.0%	1.7%	2.4%
	$n = 20$	0.5%	2.0%	3.4%	4.7%
	$n = 40$	0.7%	3.4%	5.7%	7.7%
	$n = 80$	0.8%	5.2%	8.7%	11.7%

Table 1: Cost of Multitasking in Problem $1 \mid mt \mid \sum w_jC_j$.

costs of multitasking in the unweighted problem $1 \mid mt \mid \sum C_j$ are on average 26.2% less than those in the weighted problem. This is because, in the weighted problem, the jobs with larger weights tend to be scheduled in the front part of the schedule, which is the part that is most delayed by multitasking since more jobs are waiting.

8.2 Maximum lateness

Next, we consider the maximum lateness problem, $1 \mid mt \mid L_{\max}$. Let $L'_{\max}$ and $L_{\max}$ denote the maximum lateness of problem $1 \mid mt \mid L_{\max}$ and problem $1 \mid\mid L_{\max}$, respectively. The parameter settings used are the same as those in Section 8.1, and we generate the due dates of jobs as follows. Denote $d_0 = 0$. For $j = 1, 2, \ldots, n$, we set $d_j = d_{j-1} + X_j$ with X_j being uniformly distributed in $\{50, 51, \ldots, 200\}$. Hall and Posner (2001) provide a justification of this approach and a discussion of various due date generation methods.

We let $avg(L'_{\max} - L_{\max})$ denote the average value of $L'_{\max} - L_{\max}$ over the 300 test instances for each $(n, 0, c, D)$ quadruple, and let $avg(L_{\max})$ denote the average value of $L_{\max}$ over these instances. The computational results over the 4,800 instances described in Section 8.1 are summarized in Ta-

ble 2. The quantity $avg(L'_{\max} - L_{\max})/avg(L_{\max}) \times 100\%$ is used as an estimate of the multitasking cost or value, expressed as a percentage of the expected maximum lateness of the classical model. We use $avg(L'_{\max} - L_{\max})/avg(L_{\max}) \times 100\%$ instead of $avg((L'_{\max} - L_{\max})/L_{\max}) \times 100\%$ to estimate the multitasking cost or value, since $L_{\max}$ may be zero or negative in some test instances.

		$avg(L'_{\max} - L_{\max})/avg(L_{\max}) \times 100\%$			
		$(0, c, D) = (0, 0.1, 0.005)$	$(0, c, D) = (0, 0.2, 0.01)$	$(0, c, D) = (0, 0.3, 0.015)$	$(0, c, D) = (0, 0.4, 0.02)$
$f(k) = 0.1 \times k$	$n = 10$	5.4%	8.0%	10.6%	13.3%
	$n = 20$	14.3%	21.9%	29.6%	37.6%
	$n = 40$	38.3%	59.2%	81.2%	103.0%
	$n = 80$	109.3%	168.0%	224.7%	277.2%
$f(k) = -0.1 \times k$	$n = 10$	−0.4%	2.2%	4.8%	7.5%
	$n = 20$	−0.1%	7.4%	15.0%	23.0%
	$n = 40$	−0.6%	19.2%	40.4%	62.0%
	$n = 80$	3.1%	61.0%	118.5%	172.7%

Table 2: Cost of Multitasking in Problem $1 \mid mt \mid L_{\max}$.

The results for the maximum lateness problem in Table 2 show that the increase in multitasking cost for larger instances is more than proportional to the number of jobs. This indicates that the cost of multitasking is sensitive to a change in problem size when the scheduling objective is to minimize job lateness. When there is a value of interruption, the results for low c and D parameter values demonstrate that a small net benefit from multitasking is possible when the number of jobs is low. However, this result does not continue as the number of jobs increases, and for 80 jobs the net cost of multitasking ranges from 3.1% to 172.7%. Increases in the c and D parameter values produce a less than proportionate increase in the cost of multitasking, but a more than proportionate increase in the net cost, as a result of the relatively significant value of multitasking at low parameter values. On average, the value of interruption reduces the net cost of multitasking by 55.4%.

8.3 Weighted number of late jobs

We consider the weighted number of late jobs problem, $1 \mid mt \mid \sum w_j U_j$. Since the problem is binary *NP*-hard for the unweighted case and unary *NP*-hard for the weighted case, we consider only the special cases where $g_i(p'_c) = Dp'_i$ or $g_i(p'_i) = c$. The unweighted version of these special cases can be solved by Algorithm U in Section 6, and the weighted version of these special cases can be solved

by Algorithms WU1 and WU2 in Section 7. We conduct two separate experiments, one for the case where $g_i(p_i') = c$ and the other for the case where $g_i(p_i') = Dp_i'$.

The parameter settings in these experiments are the same as those in Section 8.2. Let U_j' and U_j denote a binary variable that indicates whether job j is late in problem $1 \mid mt \mid \sum w_j U_j$ and problem $1 \mid\mid \sum w_j U_j$, respectively. As in Section 8.2, the quantity $avg(\sum w_j U_j' - \sum w_j U_j)/avg(\sum w_j U_j) \times 100\%$ is used as an estimate of the multitasking cost or value.

Table 3 presents results for both the weighted and unweighted number of late jobs problems. We first discuss the weighted problem. When there is no value of interruption and $g_i(p_i') = Dp_i'$, the average cost of multitasking ranges from 7.3% to 33.5% for 10 jobs, and from 241.4% to 1021.9% for 80 jobs. The corresponding results when $g_i(p_i') = c$ are from 2.4% to 4.8% for 10 jobs, and from 61.5% to 164.4% for 80 jobs. In both cases, the increase in multitasking cost for larger instances is more than proportional to the number of jobs. On average, the value of interruption reduces the net cost of multitasking by 29.6%.

The average net cost of multitasking in the unweighted problem $1 \mid mt \mid \sum U_j$ is 19.3% less than that in the weighted problem, and this difference is greater under the interruption function $g_i(p_i') = Dp_i'$. In the weighted problem, job weight and processing time are both important criteria for deciding whether a job should be on-time or late. In the presence of multitasking, the processing time becomes a more important criterion, because job interruptions make it harder for the primary jobs to meet their due dates, hence less priority is given to selecting jobs with smaller weights to be late jobs. This increases the average cost of multitasking in the weighted case. Moreover, for the interruption function $g_i(p_i') = Dp_i'$, the interruption time depends on the lengths of the waiting jobs, which strengthens this effect.

9 Concluding Remarks

This paper studies the impact of the widely observed behavioral phenomenon of multitasking on a simple scheduling environment. We provide an illustrative case study, based on an administrative planning scenario. We consider several of the most important practical scheduling problems, and for each, where possible, describe an efficient optimal algorithm that allows for multitasking. We show that one of these efficiently solvable problems, minimizing the number of late jobs, becomes intractable in the presence of multitasking. Also, for the problem of minimizing the weighted

		$avg(\sum w_j U'_j - \sum w_j U_j)/avg(\sum w_j U_j) \times 100\%$			
		$D = 0.005$	$D = 0.01$	$D = 0.015$	$D = 0.02$
Weighted problem; $f(k) = 0.1 \times k$	$n = 10$	7.3%	15.1%	24.8%	33.5%
	$n = 20$	23.5%	45.2%	68.3%	89.9%
	$n = 40$	70.9%	139.5%	216.8%	292.6%
	$n = 80$	241.4%	502.5%	769.5%	1021.9%
Weighted problem; $f(k) = -0.1 \times k$	$n = 10$	4.3%	12.5%	19.4%	30.0%
	$n = 20$	14.3%	37.7%	55.9%	79.3%
	$n = 40$	41.1%	104.3%	175.8%	252.2%
	$n = 80$	131.9%	365.1%	616.1%	862.1%
		$c = 0.1$	$c = 0.2$	$c = 0.3$	$c = 0.4$
Weighted problem; $f(k) = 0.1 \times k$	$n = 10$	2.4%	3.2%	4.1%	4.8%
	$n = 20$	6.3%	9.8%	13.0%	16.2%
	$n = 40$	20.5%	27.7%	37.7%	49.3%
	$n = 80$	61.5%	93.1%	125.8%	164.4%
Weighted problem; $f(k) = -0.1 \times k$	$n = 10$	−0.2%	1.4%	2.2%	2.8%
	$n = 20$	−0.6%	1.8%	3.9%	7.6%
	$n = 40$	−3.0%	3.3%	12.2%	20.2%
	$n = 80$	−9.9%	11.2%	36.6%	66.8%
		$D = 0.005$	$D = 0.01$	$D = 0.015$	$D = 0.02$
Unweighted problem; $f(k) = 0.1 \times k$	$n = 10$	6.1%	12.3%	20.9%	27.7%
	$n = 20$	22.6%	43.2%	61.6%	82.9%
	$n = 40$	68.6%	127.2%	191.5%	252.8%
	$n = 80$	213.5%	400.5%	576.5%	737.5%
Unweighted problem; $f(k) = -0.1 \times k$	$n = 10$	3.9%	9.5%	15.9%	24.3%
	$n = 20$	12.7%	32.7%	50.9%	70.8%
	$n = 40$	36.2%	93.9%	154.3%	216.3%
	$n = 80$	111.9%	290.4%	461.4%	623.0%
		$c = 0.1$	$c = 0.2$	$c = 0.3$	$c = 0.4$
Unweighted problem; $f(k) = 0.1 \times k$	$n = 10$	1.7%	3.1%	3.6%	4.5%
	$n = 20$	6.4%	9.4%	11.8%	15.6%
	$n = 40$	22.9%	29.7%	37.9%	47.7%
	$n = 80$	67.7%	91.6%	120.2%	149.1%
Unweighted problem; $f(k) = -0.1 \times k$	$n = 10$	−0.8%	−0.3%	0.8%	2.5%
	$n = 20$	−2.0%	0.7%	2.4%	5.9%
	$n = 40$	−5.5%	0.3%	9.0%	16.9%
	$n = 80$	−18.0%	4.1%	28.2%	54.4%

Table 3: Cost of Multitasking in Problem 1 | mt | $\sum w_j U_j$.

number of late jobs, there are pseudo-polynomial time algorithms for the classical scheduling model, and yet no such algorithm exists in the presence of multitasking, unless $P = NP$. Finally, we

investigate the impact of multitasking on the total scheduling cost, by means of a computational study. Our results show that the cost of multitasking is significant for most combinations of instance size and parameter settings. Hence, it may be worthwhile for companies to invest in systems or processes to reduce multitasking.

Our work provides several insights to decision makers in administrative, manufacturing, and process management applications. First, multitasking can create significant and costly interruptions in scheduling problems. Second, several scheduling problems with multitasking do not respond well to the use of solution procedures designed for the corresponding classical problems. Hence, new solution procedures are needed, especially if finding an optimal solution is important. Third, in some cases, optimal solutions are harder to find than for the corresponding classical model, which motivates the design and evaluation of heuristic approaches. Fourth, our results on the cost of multitasking in Section 8 enable a more precise evaluation of the tradeoff among priority, multitasking, and critical chain (Goldratt 1997) approaches to setting priorities across multiple projects. Fifth, our discussion of the cost or value of multitasking informs companies about how much it would be worthwhile to invest in systems or processes, in order to eliminate or encourage multitasking. Finally, since multitasking is an everyday phenomenon in many activities (Rosen 2008), we hope that our work will also contribute to a general understanding of the importance of focus in such activities.

A number of interesting problems remain open for further research. First, as discussed in Section 1, there are additional motivations for multitasking that should be modeled stochastically. Second, the classical scheduling literature contains a large number of problems that remain to be studied in the presence of multitasking. Third, our assumption that processing during the interruption of other jobs need not be repeated can be varied; either partial or complete repetition of processing can be motivated by practical issues. Fourth, our work assumes that no interrupting job is completed before it becomes a primary job, and it would be valuable to consider an alternative model where this assumption is relaxed. Fifth, we consider a general but not universal model of the interruption function. While our definition is apparently a natural one, there are alternatives. For example, there may be specific applications where the interruption time depends on the characteristics of both the primary job and the waiting jobs. Sixth, the effects of multitasking on quality and creativity have not been studied for business processes. Indications are that such

issues may be significant (Elder 2006, Jez 2011). Seventh, scheduling problems with multitasking represent a completely open area for the study of approximability issues, for example the design of heuristic performance analysis and approximation schemes for intractable problems (Schuurman and Woeginger 2007). Eighth, our work motivates the development and evaluation of practical measures to reduce or increase multitasking by the waiting jobs. Such measures include ensuring physical separation of the waiting jobs, imposing administrative controls on those jobs, and designing appropriate incentives that encourage focus on the scheduled job (Babauta 2007) or encourage multitasking. Ninth, our work identifies a link between operations research and behavioral psychology; apart from the well established link for queueing systems, such links are rare, and this new link should be explored further. Finally, our work motivates further research to evaluate the costs incurred by multitasking, relative to other priority approaches, in managing multiple projects. In conclusion, we hope that our work will encourage research on these important and practical issues.

References

Altmann, E.M., J.G. Trafton. 2007. Timecourse of recovery from task interruption: Data and a model. *Psychonomic Bulletin & Review* **14**(6): 1079–1084.

Babauta, L. 2007. *How NOT to Multitask—Work Simpler and Saner.* Available at http://zenhabits.net/how-not-to-multitask-work-simpler-and/ (accessed date November 18, 2014).

Brucker, P. 2007. *Scheduling Algorithms,* 5th edition. Springer, Berlin.

Cantor, J. 2010. Five reasons we multitask anyway. *Psychology Today: Conquering Cyber Overload,* May 31. Available at http://www.psychologytoday.com/blog/conquering-cyber-overload (accessed date November 18, 2014).

Chisholm, C.D., E.K. Collison, D.R. Nelson, W.H. Cordell. 2000. Emergency department workplace interruptions: Are emergency physicians "interrupt-driven" and "multitasking"? *Academic Emergency Medicine* **7**(11): 1239–1243.

Coffman, E.G., R.R. Muntz, H. Trotter. 1970. Waiting time distributions for processor-sharing systems. *Journal of the Association for Computing Machinery* **17**(1): 123–130.

Craig, A. 1985. Field studies of human inspection: The application of vigilance research. S. Folkard, T.H. Monk, eds. *Hours of Work: Temporal Factors in Work-Scheduling,* Chapter 12. Wiley,

Chichester, 133–145.

Czerwinski, M., E. Horvitz, S. Wilhite. 2004. A diary study of task switching and interruptions. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04), pp. 175–182.

Damian, I. 2009. Multitasking: Working beyond your limits. *Free Mind and Speech,* January 29. Available at http://freemindandspeech.blogspot.com (accessed date November 18, 2014).

Dobson, G., T. Tezcan, V. Tilson. 2013. Optimal workflow decisions for investigators in systems with interruptions. *Management Science* **59**(5): 1125–1141.

Elder, A. 2006. The five diseases of project management. White paper, No Limits Leadership, Inc. Available at http://www.nolimitsleadership.com/images/The%20Five%20Diseases%20of%20Project%20Management.pdf (accessed date November 18, 2014).

Fuhrmann, S.W., R.B. Cooper. 1985. Stochastic decompositions in the $M/G/1$ queue with generalized vacations. *Operations Research* **33**(5): 1117–1129.

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

Get More YouTube Views. 2010. *Multitasking Effects Revealed.* Available at http://www.getmoreyoutubeviews.com/2010/04/multitasking-effects-revealed/ (accessed date November 18, 2014).

Goldratt, E.M. 1997. *The Critical Chain.* The North River Press, Great Barrington, MA.

Gonzalez, M.J. 1977. Deterministic processor scheduling. *ACM Computing Surveys* **9**(3): 173–204.

González, V.M., G. Mark. 2004. "Constant, constant, multi-tasking craziness": Managing multiple working spheres. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04), pp. 113–120.

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

Grundgeiger, T., P. Sanderson, H.G. MacDougall, B. Venkatesh. 2010. Interruption management in the intensive care unit: Predicting resumption times and assessing distributed support. *Journal of Experimental Psychology: Applied* **16**(4): 317–334.

Hall, N.G., M.E. Posner. 2001. Generating experimental data for computational testing with machine scheduling applications. *Operations Research* **49**(7): 854–865.

Huff, C. 2007. Focus. *American Way,* November 1, 34–36.

Iqbal, S.T., B.P. Bailey. 2006. Leveraging characteristics of task structure to predict the cost of interruption. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06), pp. 741–750.

Iqbal, S.T., E. Horvitz. 2007. Disruption and recovery of computing tasks: Field study, analysis, and directions. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07), pp. 677–686.

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

Järrehult, B. 2012. Can multi-tasking result in more than 60% longer project time? Available at: http://www.innovationmanagement.se/2012/08/14/can-multi-tasking-result-in-more-than-60-longer-project-time/ (accessed date November 18, 2014).

Jett, Q.R., J.M. George. 2003. Work interrupted: A closer look at the role of interruptions in organizational life. *Academy of Management Review* **28**(3): 494–507.

Jez, V. 2011. Searching for the meaning of multitasking. Norsk konferanse for organisasjoners bruk av informasjonsteknologi (NOKOBIT 2011), pp. 157–166.

Karp, R.M. 1972. Reducibility among combinatorial problems. R.E. Miller, J.W. Thatcher, eds. *Complexity of Computer Computations.* Plenum Press, New York, 85–103.

Kerzner, H.R. 2013. *Project Management: A Systems Approach to Planning, Scheduling, and Controlling,* 11th edition. Wiley, Hoboken, NJ.

Klastorin, T. 2004. *Project Management: Tools and Trade-Offs.* Wiley, Hoboken, NJ.

Lawler, E.L., J.M. Moore. 1969. A functional equation and its application to resource allocation and sequencing problems. *Management Science* **16**(1): 77–84.

Leach, L.P. 1999. Critical chain project management improves project performance. *Project Management Journal* **30**(2): 39–51.

Lee, C.-Y. 1996. Machine scheduling with an availability constraint. *Journal of Global Optimization* **9**(3–4): 395–416.

Loeb, M., E.A. Alluisi. 1977. An update of findings regarding vigilance and a reconsideration of underlying mechanisms. R.R. Mackie, ed. *Vigilance: Theory, Operational Performance, and Physiological Correlates.* Plenum Press, New York, 719–749.

Ma, Y., C. Chu, C. Zuo. 2010. A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering* **58**(2): 199–211.

Merriam-Webster Online. 2014. *Multitasking.* Merriam-Webster, Incorporated. Available at: http://www.merriam-webster.com/dictionary/multitasking (accessed date November 18, 2014).

Moore, J.M. 1968. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science* **15**(1): 102–109.

Morgenstern, J. 2004. *Making Work Work: New Strategies for Surviving and Thriving at the Office.* Simon & Schuster, New York.

O'Leary, K.J., D.M. Liebovitz, D.W. Baker. 2006. How hospitalists spend their time: Insights on efficiency and safety. *Journal of Hospital Medicine* **1**(2): 88–93.

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

Rand, G.K. 2000. Critical chain: the theory of constraints applied to project management. *International Journal of Project Management* **18**(3): 173–177.

Realization. 2014. The effects of multitasking on organizations. Available at: http://www.realization.com/pdf/Effects_of_Multitasking_on_Organizations.pdf (accessed date November 18, 2014).

Rosen, C. 2008. The myth of multitasking. *The New Atlantis* **20**: 105–110.

Rubinstein, J.S., D.E. Meyer, J.E. Evans. 2001. Executive control of cognitive processes in task switching. *Journal of Experimental Psychology: Human Perception and Performance* **27**(4): 763–797.

Salvucci, D.D., N.A. Taatgen. 2011. *The Multitasking Mind.* Oxford University Press, New York.

Schuurman, P., G.J. Woeginger. 2007. Approximation schemes—A tutorial. Working paper, Technical University of Eindhoven, The Netherlands. Available at http://www.win.tue.nl/∼gwoegi/papers/ptas.pdf (accessed date November 18, 2014).

Seshadri, S., Z. Shapira. 2001. Managerial allocation of time and effort: The effects of interruptions. *Management Science* **47**(5): 647–662.

Smith, W.E. 1956. Various optimizers for single-stage production. *Naval Research Logistics Quarterly* **3**(1–2): 59–66.

Speier, C., J.S. Valacich, I. Vessey. 1999. The influence of task interruption on individual decision making: An information overload perspective. *Decision Sciences* **30**(2): 337–360.

Stanhope, P.D. 1847. *The Letters of Philip Dormer Stanhope, Earl of Chesterfield.* Ed. Lord Mahon. Richard Bentley, London.

Suddath, C. 2012. My life as an efficiency squirrel. *Bloomberg Businessweek,* October 29 – November 4, **4302**: 88–89.

theSalmonFarm. 2007. Guest blogger on "I don't have a poor attention span! I'm multitasking." Available at: http://thesalmonfarm.org/blog/p/370 (accessed date November 18, 2014)

Ware, R., R.A. Baker. 1977. The effect of mental set and states of consciousness on vigilance decrement: A systematic exploration. R.R. Mackie, ed. *Vigilance: Theory, Operational Performance, and Physiological Correlates.* Plenum Press, New York, 603–616.