Discrete Applied Mathematics 208 (2016) 41–58

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Multitasking via alternate and shared processing: Algorithms and complexity

Nicholas G. Hall[a], Joseph Y.-T. Leung[b], Chung-Lun Li[c,*]

[a] Fisher College of Business, The Ohio State University, Columbus, OH 43210, United States

[b] Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, United States

[c] Department of Logistics and Maritime Studies, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

ARTICLE INFO

Article history:
Received 30 September 2014
Received in revised form 17 March 2016
Accepted 29 March 2016
Available online 25 April 2016

Keywords:
Scheduling
Motivations for multitasking
Efficient algorithm
Intractability

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.

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

* Corresponding author.
E-mail addresses: hall.33@osu.edu (N.G. Hall), leung@cis.njit.edu (J.Y.-T. Leung), chung-lun.li@polyu.edu.hk (C.-L. Li).

http://dx.doi.org/10.1016/j.dam.2016.03.018
0166-218X/© 2016 Elsevier B.V. All rights reserved.

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-aware 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 $[\tau_o, \tau_o + \tau_e]$, $[2\tau_o + \tau_e, 2(\tau_o + \tau_e)]$, $[2(\tau_o + \tau_e) + \tau_o, 3(\tau_o + \tau_e)]$, ..., and the second machine is unavailable during periods $[0, \tau_o]$, $[\tau_o + \tau_e, 2\tau_o + \tau_e]$, $[2(\tau_o + \tau_e), 2(\tau_o + \tau_e) + \tau_o]$, ... 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 [2,9,11,14,23], 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 $L_{\max} = \max_{1 \leq j \leq n}\{L_j\}$, 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.