DISCRETE
APPLIED
MATHEMATICS

Parallel machine scheduling with a common server

Nicholas G. Hall[a, *], Chris N. Potts[b], Chelliah Sriskandarajah[c]

[a] *Department of Management Sciences, The Ohio State University, 2100, Neil Avenue, Columbus, OH 43210-1144, USA*

[b] *Faculty of Mathematical Studies, University of Southampton, High Field, Southampton, 5O9 5NH, UK*

[c] *School of Management, The University of Texas at Dallas, 2601 N. Floyd Road, Richardson, TX 75083-06888, USA*

Received 15 August 1997; revised 16 April 1999; accepted 28 June 1999

Abstract

This paper considers the nonpreemptive scheduling of a given set of jobs on several identical, parallel machines. Each job must be processed on one of the machines. Prior to processing, a job must be loaded (setup) by a single server onto the relevant machine. The paper considers a number of classical scheduling objectives in this environment, under a variety of assumptions about setup and processing times. For each problem considered, the intention is to provide either a polynomial- or pseudo-polynomial-time algorithm, or a proof of binary or unary NP-completeness. The results provide a mapping of the computational complexity of these problems. © 2000 Elsevier Science B.V. All rights reserved.

MSC: 90B35

Keywords: Deterministic parallel machine scheduling; Common server; Polynomial-time algorithm; NP-complete

1. Introduction

We consider a deterministic scheduling environment with m identical, parallel machines. Each of n given jobs must be processed nonpreemptively on one of the machines. The loading of a job on a machine, which we call a *setup*, is performed by a single server, and cannot occur while the machine is processing a job. However, machines can process jobs unattended. Therefore, having completed a setup, the server is free to perform other setup operations. Simultaneous requests by machines for the server will necessarily result in machine idle time.

In this environment, we consider a variety of possibilities for the setup time. Specifically, this can require unit time for all jobs, or an equal but arbitrary time for all jobs,

* Corresponding author.
E-mail address: halln@cob.ohio-state.edu (N.G. Hall)

0166-218X/00/$ - see front matter
PII: S0166-218X(99)00206-1

or can vary between jobs. Problems with unit processing times, and with processing times that vary between jobs, are analyzed. A variety of classical scheduling objectives, including the minimization of makespan, maximum lateness, total completion time, and the number of late jobs, as well as weighted versions of some of these, are considered.

The type of server varies according to the production environment. The setup or loading of a machine may be undertaken by a human operator. In modern manufacturing systems, however, it would be usual for a robot or an automated guided vehicle (AGV) to perform the setups. More generally, some resource such as a piece of equipment may be required throughout the setup process. Each of these situations defines a scheduling problem with a common server. The reasons for the use of only a single, common server include cost, physical space limitations and the need to simplify the production process.

Useful general references on machine scheduling include the works of Conway et al. [3], and Baker [2]. Of particular relevance to our study is the machine interference problem [1,27,28,35]. This problem involves finding the optimal number of machines to be assigned to a single operator, in order to minimize operator interference, which occurs when several machines simultaneously request the operator's services.

There have been numerous studies of robots [32] and AGVs [6] in different scheduling environments. Inaba and Sakakibara [15], Hartley [13], and Hull [14] discuss several applications in which the use of robots has significantly enhanced productivity. Wilhelm and Sarin [36] study the computational complexity of several robot-served manufacturing environments. Hall et al. [10,11], Kamoun et al. [17] and Sriskandarajah et al. [34] provide many theoretical results for scheduling in a robotic cell, which is equivalent to a flowshop with a single robot server and no intermediate buffers.

Heuristics for parallel machine scheduling with a common server have been the subject of four studies. Koulamas and Smith [20] propose a look ahead heuristic for an environment with continuously arriving jobs. Koulamas [19] proposes a beam search heuristic for a static environment with two machines. So far as we are aware, there is no previous study of algorithmic and computational complexity issues in this environment. Kravchenko and Werner [21] subsequently resolve the complexity of one open problem from this paper, and discuss some related problems. Also in subsequent work, Glass et al. [8] consider related models with parallel machines to which jobs are dedicated, and provide algorithmic, complexity and heuristic analysis results.

Our objective is to provide a map of the computational complexity of these problems. Thus, we attempt to provide an efficient algorithm, or prove a computational complexity result which shows that such an algorithm is unlikely to exist. Some of these results are obtained by adapting the classical algorithms of Jackson [16], Smith [33], Lawler [22–24], Moore [30], Emmons [5], Lawler and Moore [25] and Monma [29] to the problem of scheduling the common server.

In Section 2, we describe our notation and classification scheme, define two useful scheduling concepts, and provide an overview of the algorithmic and computational

complexity results in the paper. Section 3 analyzes the relationship between problems with unit processing times and classical single-machine scheduling problems. Sections 4–8 describe the algorithmic and computational complexity results in detail, organized by objective function. Finally, Section 9 contains some conclusions and suggestions for future research.

2. Preliminaries

Here we summarize our notation and problem classification scheme, define the useful concepts of active and list schedules, and provide an overview of the results in the paper.

2.1. Notation and classification

We begin with some notation. Let $N = \{1, \ldots, n\}$ denote the set of jobs. There are m identical parallel machines, and we let M_i denote the ith machine. Let p_j denote the processing time, d_j the nonnegative due date, and w_j the nonnegative weight (or value), of job j for $j \in N$. Also, let s_j denote the time for the server to perform a setup for job j. No interruption in a setup or in the processsing of a job is allowed. We assume throughout that all p_j and s_j are integer, for $j \in N$. In the derivation of algorithms, we assume that $p_j \geqslant 1$ and $s_j \geqslant 1$, for $j \in N$. Some of the computational complexity proofs in the paper allow $p_j = 0$. However, this is permitted for clarity of exposition only, and can be avoided by rescaling.

We also define the following measures for each job j of any schedule σ:

$$C_j(\sigma) = \text{the time at which job } j \text{ is completed},$$

$$L_j(\sigma) = C_j(\sigma) - d_j, \text{ the lateness of job } j,$$

$$U_j(\sigma) = \begin{cases} 1 \text{ if job } j \text{ is completed by its due date,} \\ 0 \text{ otherwise,} \end{cases}$$

$$T_j(\sigma) = \max\{C_j(\sigma) - d_j, 0\}, \text{ the tardiness of job } j.$$

Where no ambiguity arises, we simplify $C_j(\sigma)$ to C_j, $L_j(\sigma)$ to L_j, $U_j(\sigma)$ to U_j, and $T_j(\sigma)$ to T_j.

The standard classification scheme for scheduling problems [9] is $\alpha|\beta|\gamma$, where α indicates the scheduling environment, β describes the job characteristics or restrictive requirements, and γ defines the objective function to be minimized. Our classification scheme is similar, but it also allows for the existence of a single server (S) in α. Thus, $\alpha = Pm, S$ if there are a fixed number, m, of identical parallel machines; otherwise, $\alpha = P, S$ indicates that the number of machines is arbitrary. Under β, we use $p_j = 1$ to denote unit processing times, $s_j = 1$ to denote unit setup times, and $s_j = s$ to denote that all setup times are equal to an arbitrary, positive integer s. The default assumption is that processing and setup times are arbitrary.

The objective functions that we consider are as follows:

$C_{\max} = \max_{j=1,\ldots,n} \{C_j\}$, the latest completion time of any job, or the makespan,

$L_{\max} = \max_{j=1,\ldots,n} \{C_j - d_j\}$, the maximum lateness of any job,

$\sum_{j=1}^{n} C_j =$ the total completion time of jobs,

$\sum_{j=1}^{n} w_j C_j =$ the total weighted completion time of jobs,

$\sum_{j=1}^{n} U_j =$ the number of jobs not completed by their due dates,

$\sum_{j=1}^{n} w_j U_j =$ the total weight of jobs not completed by their due dates,

$\sum_{j=1}^{n} T_j =$ the total tardiness of jobs,

$\sum_{j=1}^{n} w_j T_j =$ the total weighted tardiness of jobs.

2.2. Active and list schedules

We say that a schedule is *active* if, for a given sequence of jobs, it is not possible to reduce the completion time of any job without increasing the completion time of another job.

We will also make use of the following generic list scheduling algorithm.

Algorithm List

Initialization

Order the jobs in some sequence σ.

Schedule construction

Execute the following for $j = 1, \ldots, n$.

Add job $\sigma(j)$ to form an active schedule.

In the Schedule Construction step, job $\sigma(j)$ is assigned to the first machine, M_i, to become available. Moreover, if the server is free, the setup for $\sigma(j)$ starts when M_i completes any previous processing; otherwise, the start of the setup is delayed until the server becomes available.

We will make use of *List* at various points in the paper, with a variety of definitions for σ.

Table 1
The complexity of the problems

Objective	m	$p_j = 1$	s_j	Complexity results	
$C_{\max}$	2	No	1	Binary NPC, Open[a]	Theorem 4.1
	2	No	s	Unary NPC	Theorem 4.2
	Arb	Yes	Arb	O(1)	Theorem 4.3
$L_{\max}$	2	No	1	Unary NPC	Theorem 5.1
	Arb	Yes	Arb	$O(n \log n)$	Theorem 5.2
$\sum_{j=1}^{n} C_j$	2	No	1	$O(n \log n)$	Theorem 6.1
	2	No	s	Unary NPC	Theorem 6.2
	Arb	No	Any	Open	
	Arb	Yes	s	O(1)	Theorem 6.3
	Arb	Yes	Arb	$O(n \log n)$	Theorem 6.4
$\sum_{j=1}^{n} w_j C_j$	2	No	1	Binary NPC, Open	Theorem 6.5
	Arb	Yes	Arb	$O(n \log n)$	Theorem 6.6
$\sum_{j=1}^{n} U_j$	2	No	1	Unary NPC	Corollary 7.1
	Arb	Yes	s	$O(n)$	Table 2
	Arb	Yes	Arb	$O(n \log n)$	Table 2
$\sum_{j=1}^{n} w_j U_j$	Arb	Yes	s	$O(n \log n)$	Table 2
	Arb[b]	Yes	Arb	Binary NPC, $O(n \sum_{j=1}^{n} s_j)$	Table 2
$\sum_{j=1}^{n} T_j$	2	No	1	Unary NPC	Corollary 8.1
	Arb	Yes	s	$O(n \log n)$	Table 3
	Arb[b]	Yes	Arb	Binary NPC, $O(n^4 \sum_{j=1}^{n} s_j)$	Table 3
$\sum_{j=1}^{n} w_j T_j$	Arb	Yes	s	$O(n^3)$	Table 3
	2	Yes	Arb	Unary NPC	Table 3

[a] Kravchenko and Werner [21] subsequently describe a pseudo-polynomial-time algorithm.
[b] Binary NPC even for $m = 2$.

2.3. Overview of the results

Table 1 presents the results of this paper. We use Unary (respectively, Binary) NPC to indicate that the equivalent recognition version of a problem is NP-complete with respect to a unary (resp., binary) encoding of the data. Related definitions can be found in [7,31]. The second column indicates either a fixed number of machines, or the fact that the number of machines is arbitrary (denoted by "Arb"). A "No" in the third column indicates that processing times may vary between jobs, and a "Yes" indicates that all processing times are one unit. A "1" in the fourth column shows that all setup times are one unit, an "s" shows that all setup times are equal but their length may vary with problem input, "Arb" shows that setup times are arbitrary, and "Any" indicates all three of these possibilities. Included in each complexity result cell is a reference to where a proof of that result can be found.

Note that a problem with unit setup times is a special case of the corresponding problem with all setup times equal to a positive integer s, which in turn is a special case of the corresponding problem with arbitrary setup times. When stating an NP-completeness result, we assume that the corresponding result also holds for its

generalizations. Similarly, an algorithm for a more general problem is also applicable to a special case.

3. Unit processing times

In this section, we provide links between problem P, $S|p_j = 1|\gamma$, where γ is one of the objective functions introduced in Section 2.1, and the corresponding classical single-machine problem $1||\gamma'$. For problem $1||\gamma'$, we let $N' = \{1,\ldots,n'\}$ denote the set of jobs, and p'_j the processing time of job j for $j \in N'$. Moreover, if applicable, the due date and weight of job j, for $j \in N'$, are denoted by d'_j and w'_j, respectively.

Theorem 3.1. *Problem* P, $S|p_j = 1|\gamma$ (*respectively*, P, $S|p_j = 1, s_j = s|\gamma$), *where* $\gamma \in \{C_{\max}, \sum_{j=1}^{n} C_j, \sum_{j=1}^{n} w_j C_j\}$, *is equivalent to problem* $1||\gamma'$ *with* $n' = n$, *and* $p'_j = s_j$ *and* (*if applicable*) $w'_j = w_j$, *for* $j = 1,\ldots,n$, (*resp.* $1|p'_j = p'|\gamma'$ *with* $n' = n$, *and* $p'_j = p' = s$ *and* (*if applicable*) $w'_j = w_j$, *for* $j = 1,\ldots,n$), *where* $\gamma' \in \{C_{\max}, \sum_{j=1}^{n'} C_j, \sum_{j=1}^{n'} w'_j C_j\}$.

Proof. Consider an active schedule for problem P, $S|p_j = 1|\gamma$ or P, $S|p_j = 1, s_j = s|\gamma$ in which the completion time of job j, for $j \in N$, is C_j. Since each set-up time is at least as long as any processing time, the server forms a bottleneck and is continuously busy throughout the interval $[0, \sum_{j=1}^{n} s_j]$. Let C'_j denote the completion time of the set-up for job j, for $j \in N$. Note that C'_j is the completion time of job j in problem $1||\gamma'$ or $1|p'_j = p'|\gamma'$ for an active schedule in which the processing order for the server defines the job sequence. Since $C_j = C'_j + 1$, minimizing $\max_{j\in\{1,\ldots,n\}}\{C_j\}$ is equivalent to minimizing $\max_{j\in\{1,\ldots,n'\}}\{C'_j\}$, and minimizing $\sum_{j=1}^{n}(w_j)C_j$ is equivalent to minimizing $\sum_{j=1}^{n'}(w'_j)C'_j$. □

Theorem 3.2. *Problem* P, $S|p_j = 1|\gamma$ (*respectively*, P, $S|p_j = 1$, $s_j = s|\gamma$), *where* $\gamma \in \{L_{\max}, \sum_{j=1}^{n} U_j, \sum_{j=1}^{n} w_j U_j, \sum_{j=1}^{n} T_j, \sum_{j=1}^{n} w_j T_j\}$, *is equivalent to problem* $1||\gamma'$ *with* $n' = n$, *and* $p'_j = s_j$, $d'_j = d_j - 1$ *and* (*if applicable*) $w'_j = w_j$, *for* $j = 1,\ldots,n$, (*resp.* $1|p'_j = p'|\gamma'$*with* $n' = n$), *and* $p'_j = p' = s$, $d'_j = d_j - 1$ *and* (*if applicable*) $w'_j = w_j$, *for* $j = 1,\ldots,n$, *where* $\gamma' \in \{L_{\max}, \sum_{j=1}^{n'} U_j, \sum_{j=1}^{n'} w'_j U_j, \sum_{j=1}^{n'} T_j, \sum_{j=1}^{n'} w'_j T_j\}$. *Moreover, the objective function values for the two problems are identical.*

Proof. As in the proof of Theorem 3.1, there is a one-to-one correspondence between active schedules for problems P, $S|p_j = 1|\gamma$ and $1||\gamma'$, and for problems P, $S|p_j = 1$, $s_j = s|\gamma$ and $1|p'_j = p'|\gamma'$. As before, $C_j = C'_j + 1$, where C_j and C'_j, respectively, denote the completion times of job j in these two schedules. Since $C_j - d_j = C'_j - d'_j$, the objective function values in the two problems are identical. □

In view of Theorems 3.1 and 3.2, an algorithm for a classical problem $1||\gamma'$ or $1|p'_j = p'|\gamma'$ provides an algorithm with the same time complexity for the corresponding problem P, $S|p_j = 1|\gamma$ or P, $S|p_j = 1, s_j = s|\gamma$. Similarly, a proof of NP-completeness of

the decision version of problem $1|\,|\gamma'$ or $1|\,p'_j = p'|\gamma'$ implies that the decision version of the corresponding problem P, $S|\,p_j=1|\gamma$ or P, $S|\,p_j=1, s_j = s|\gamma$ is also NP-complete. We use these observations extensively in the subsequent sections.

4. Makespan problems

We begin by showing that problem $P2$, $S|s_j = 1|C_{\max}$ is intractable.

Theorem 4.1. *The recognition version of problem* $P2$, $S|s_j = 1|C_{\max}$ *is binary* NP-*complete.*

Proof. Consider the following problem which is known to be binary NP-complete.

Partition [7]: given r elements with integer sizes $a_1,\ldots,a_r$, does there exist a partition of the index set $\{1,\ldots,r\}$ into subsets A_1 and A_2, such that $\sum_{i\in A_1} a_i = \sum_{i\in A_2} a_i$?

Given an arbitrary instance of Partition, consider the following instance of $P2$, $S|s_j = 1|C_{\max}$:

$$n = r,$$

$$p_j = 4na_j, \quad j = 1,\ldots,r,$$

$$C = 2n - 1 + 2n\sum_{j=1}^{r} a_j,$$

where C is the threshold cost.

We now show that there exists a solution to the scheduling problem with $C_{\max} \leqslant C$ if and only if there exists a solution to Partition.

($\Rightarrow$) If there exists a solution A_1, A_2 to Partition, then the jobs corresponding to indices in A_1 can be scheduled on machine M_1, and the other jobs on machine M_2. The total processing time on each machine is $2n\sum_{j=1}^{n} a_j$. Note that no more than $n-1$ jobs are processed on either machine in an optimal schedule. Therefore, on each machine, the total setup time is at most $n-1$, and the possibility of one unit of machine idle time due to the unavailability of the server occurs at most $n-1$ times. Thus, $C_{\max} \leqslant 2n\sum_{j=1}^{n} a_j + (n-1) + (n-1) < C$.

($\Leftarrow$) Since the processing time of each job is a multiple of $4n$, the non-existence of a solution to Partition implies that the sum of processing times on one machine must be at least $4n(\sum_{j=1}^{n} a_j/2 + 1/2) > C$. Therefore, if there exists a solution to the scheduling problem with $C_{\max} \leqslant C$, then there exists a solution to Partition. □

The pseudo-polynomial-time solvability of this problem is discussed in Section 9.

Theorem 4.2. *The recognition version of problem* $P2$, $S|s_j = s|C_{\max}$ *is unary* NP-*complete.*

Proof. Consider the following problem, which is known to be unary NP-complete. *Numerical Matching with Target Sums* [7]: given three sets $X=\{x_1,\ldots,x_r\}, Y=\{y_1,\ldots,y_r\}$ and $Z=\{z_1,\ldots,z_r\}$ of positive integers, where $\sum_{i=1}^{r} z_i=\sum_{i=1}^{r} x_i+\sum_{i=1}^{r} y_i$, do there exist permutations $(j_1,\ldots,j_r)$ and $(k_1,\ldots,k_r)$ of the indices $1,\ldots,r$ such that $z_i=x_{j_i}+y_{k_i}$ for $i=1,\ldots,r$?

We assume as part of this definition that r is even, and that if there does exist a solution to Numerical Matching with Target Sums, then the jobs are indexed such that $z_i=x_i+y_i$ for $i=1,\ldots,r$.

Given an arbitrary instance of Numerical Matching with Target Sums, consider the following instance of $P2,\ S|s_j=s|C_{\max}$:

$$
\begin{aligned}
&n=3r+1,\\
&s_j=K^2,\ p_j=K+x_j, && j=1,\ldots,r,\\
&s_j=K^2,\ p_j=2K+y_{j-r}, && j=r+1,\ldots,2r,\\
&s_j=K^2,\ p_j=3K^2+3K+z_{j-2r}, && j=2r+1,\ldots,3r,\\
&s_j=K^2,\ p_j=0, && j=3r+1,\\
&C=(3r+1)K^2+3rK+\sum_{j=1}^{r} z_j,
\end{aligned}
$$

where $K=\sum_{j=1}^{r}(x_j+y_j+z_j)+1$.

We now show that there exists a solution to this instance of the scheduling problem with $C_{\max}\leqslant C$ if and only if there exists a solution to Numerical Matching with Target Sums.

($\Rightarrow$) Consider the schedule in Fig. 1, where setup times and processing times are shown for each job, and machine M_1 is idle in the interval $[0,K^2]$. While one machine processes job $2r+j$, for $j=1,\ldots,r-1$, the other machine performs a setup and processing for jobs j and $r+j$ and performs a setup for job $2r+j+1$. Thus, $C_{\max}=s_{2r+1}+\sum_{j=2r+1}^{3r} p_j\leqslant C$.

($\Leftarrow$) We note that either machine M_1 or machine M_2 must begin with K^2 units of idle time. Adding this unavoidable idle time to all the setup and processing times of jobs gives a total time of $(3r+1)K^2+3rK^2+6rK+\sum_{j=1}^{r} x_j+\sum_{j=1}^{r} y_j+\sum_{j=1}^{r} z_j+K^2=2C$. Since there are only two machines, it follows that the only idle time possible in a schedule with $C_{\max}\leqslant C$ is the unavoidable idle time in the interval $[0,K^2]$ on one machine.

The simultaneous processing of jobs $i,j\in\{1,\ldots,2r\}$ is impossible, since the server would be needed for setups of both jobs i and j at the same time. Similarly, a setup cannot overlap with the processing of job $i\in\{1,\ldots,2r\}$ on the other machine, because the server would be needed for the setup either of job i or of the job that is processed on the same machine immediately after i.

Thus, during all processing of jobs in $\{1,\ldots,2r\}$ on one machine, and during all setups, the other machine must be processing jobs in $\{2r+1,\ldots,3r\}$. Note that C can be viewed as a setup time of $(3r+1)K^2$, plus a remaining time of $3rK+\sum_{i=1}^{r} x_i+\sum_{i=1}^{r} y_i$, which is equal to the total processing time of jobs in $\{1,\ldots,2r\}$. Since a pair of jobs in $\{1,\ldots,2r\}$ cannot be processed simultaneously, it follows that, whenever a setup is

not being performed, a job in $\{1,\ldots,2r\}$ must be in process. Thus, any simultaneous processing of two jobs in $\{2r+1,\ldots,3r\}$ implies that $C_{\max} > C$.

Since the total processing time of jobs in $\{2r+1,\ldots,3r\}$ is $C-K^2$, these jobs must be scheduled so that the first starts processing immediately after its setup at time K^2, while each subsequent job starts processing immediately upon completion of the previous one. Assume without loss of generality that some job k, where $k \in \{2r+1,\ldots,3r\}$, is processed first on M_2 and completes at time $4K^2+3K+z_{k-2r}$. On M_1, the first setup of some job in $N \setminus \{2r+1,\ldots,3r\}$ starts at time K^2, and a setup starts at time $3K^2+3K+z_{k-2r}$ to allow another job in $\{2r+1,\ldots,3r\}$ to start processing upon completion of job k.

Therefore, in a schedule with $C_{\max} \leqslant C$, the interval $[K^2, 3K^2+3K+z_{k-2r}]$ is exactly filled with setups and processing for jobs in $\{1,\ldots,2r\}$. This time interval is too short for setups of three or more jobs, or for the setup and processing of two jobs in $\{r+1,\ldots,2r\}$. Moreover, a single job in $\{1,\ldots,2r\}$ by itself or combined with job $3r+1$, or two jobs in $\{1,\ldots,r\}$, cannot exactly fill the interval. Therefore, some jobs i and j, where $i \in \{1,\ldots,r\}$ and $j \in \{r+1,\ldots,2r\}$, are scheduled in $[K^2, 3K^2+3K+z_{k-2r}]$. For this interval to be exactly filled, $x_i + y_{j-r} = z_{k-2r}$. Repeating this argument for each group of three jobs shows that the existence of a schedule with $C_{\max} \leqslant C$ implies the existence of a solution to Numerical Matching with Target Sums. □

We now turn our attention to unit processing time problems.

Theorem 4.3. *Problem* $P,\ S|p_j=1|C_{\max}$ *can be solved optimally in constant time by scheduling the setup and processing for job* j *in the interval* $[\sum_{i=1}^{j-1} s_i, \sum_{i=1}^{j} s_i + 1]$ *on machine* M_1 *if* j *is odd, or on machine* M_2 *if* j *is even, for* $j=1,\ldots,n$.

Proof. From the total workload of the server and the one unit of processing that occurs after all setups are complete, we deduce a lower bound on the makespan of $\sum_{j=1}^{n} s_j + 1$. Since the constructed schedule achieves this lower bound, it is optimal. □

5. Maximum lateness problems

As in the previous section, we start our analysis by considering problems with arbitrary processing times.

Theorem 5.1. *The recognition version of problem* $P2,\ S|s_j=1|L_{\max}$ *is unary* NP-*complete.*

Proof. Consider the following problem which is known to be unary NP-complete.

3-*Partition* [7]: given $3r$ elements with integer sizes $a_1,\ldots,a_{3r}$, where $\sum_{i=1}^{3r} a_i = ry$, and $y/4 < a_i < y/2$ for $i=1,\ldots,3r$, does there exist a partition $Q_1,\ldots,Q_r$ of the index set $\{1,\ldots,3r\}$ such that $|Q_j|=3$ and $\sum_{i\in Q_j} a_i = y$ for $j=1,\ldots,r$?

As part of this definition, we assume that, if there does exist a solution to 3-partition, then the elements are numbered such that $a_{3j-2}+a_{3j-1}+a_{3j}=y$ for $j=1,\ldots,r$.

Given an arbitrary instance of 3-Partition, consider the following instance of $P2,\ S|s_j=1|L_{\max}$:

$$
\begin{aligned}
&n=r(2y+1)+1,\\
&p_j=2a_j,\ d_j=r(2y+1), \quad j=1,\ldots,3r,\\
&p_j=1,\ d_j=k(2y+1)+1, \quad j=3r+k,\ k=1,\ldots,r,\\
&p_j=0,\ d_j=k(2y+1), \quad j=4r+(k-1)(2y-3)+1,\ldots,4r+k(2y-3),\\
&\quad k=1,\ldots,r,\\
&p_j=0,\ d_j=r(2y+1)+1, \quad j=r(2y+1)+1,\\
&C=0.
\end{aligned}
$$

Let $J_1=\{1,\ldots,3r\}$, $J_2=\{3r+1,\ldots,4r\}$ and $J_3=\{4r+1,\ldots,r(2y+1)\}$. Also, we refer to a job of J_3 that has due date $k(2y+1)$ as being of *type* k, for $k=1,\ldots,r$. We now show that there exists a solution to this instance of the scheduling problem with $L_{\max}\leqslant 0$ if and only if there exists a solution to 3-Partition.

($\Rightarrow$) Consider the following schedule, illustrated in Fig. 2, where job numbers are shown, and where machine M_1 is idle throughout the interval $[0,1]$.

M_1: $(4r+1,\ldots,4r+2a_1-1,2,4r+2a_1+2a_2-1,\ldots,4r+2a_1+2a_2+2a_3-3,$
$3r+1,\ldots,4r+(r-1)(2y-3)+1,\ldots,4r+(r-1)(2y-3)$
$+2a_{3r-2}-1,3r-1,4r+(r-1)(2y-3)+2a_{3r-2}+2a_{3r-1}-1,\ldots,4r$
$+(r-1)(2y-3)+2a_{3r-2}+2a_{3r-1}+2a_{3r}-3,4r)$.

M_2: $(1,4r+2a_1,\ldots,4r+2a_1+2a_2-2,3,\ldots,3r-2,4r+(r-1)(2y-3)$
$+2a_{3r-2},\ldots,4r+(r-1)(2y-3)+2a_{3r-2}+2a_{3r-1}-2,3r,r(2y+1)+1)$.

Job 2 is scheduled so that its setup is in parallel with the last unit of processing of job 1. Similarly, the setup for job 3 occurs during the last unit of processing of job 2. The $2y-3$ jobs of J_3 which are of type 1 are setup while jobs 1, 2 and 3 are processed. Also, job $3r+1$ is setup in parallel with the last unit of processing of job 3. This pattern repeats for each group of three jobs of J_1. Moreover, each job is completed either at or before its due date; therefore, $L_{\max}=0$.

($\Leftarrow$) We note that there is unavoidable idle time in the interval $[0,1]$ on one machine. The total of this idle time and all job processing and setup time is $(2ry+r+1)+(2ry+r)+1=4ry+2r+2$. Since there are two machines, a lower bound on the maximum completion time is therefore $2ry+r+1=\max_{j=1,\ldots,n}\{d_j\}$. Thus, a schedule with $L_{\max}\leqslant 0$ has no idle time on either machine after time 1.

Jobs $4r$ and $2ry+r+1$ must be scheduled last, since any other jobs in last position would be completed after their due dates. Since the total setup time of all jobs is equal to $2ry+r+1$, the server can have no idle time in a schedule with $L_{\max}\leqslant 0$. It follows that, throughout the processing of each job of J_1, the server must be performing setups. Moreover, the setups that are performed in parallel with the first $2a_j-1$ units of processing of each such job j correspond to jobs with zero processing times, while the setup which is in parallel with the final unit of processing is for a job with a strictly positive processing time, or else machine idle time results. Thus, all jobs of J_3

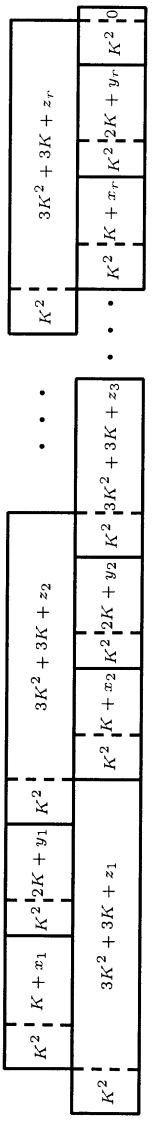
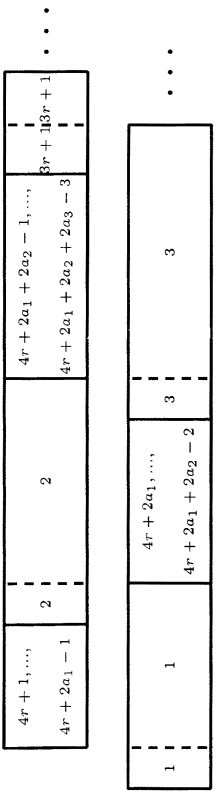


Fig. 1. A schedule with $C_{max} = C$ in Theorem 4.2.



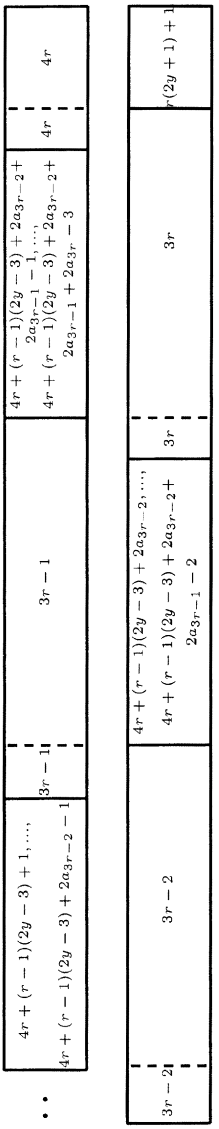


Fig. 2. A schedule with $L_{\max} = C$ in Theorem 5.1.

must be setup in parallel with the processing of jobs of J_1, but not during their last unit of processing.

Within the time interval $[0, 2y+1]$, the $2y-3$ type 1 jobs of J_3 are setup in parallel with the processing of some jobs of J_1. Since the processing time of each of these jobs is less than y and the last unit of processing time is not performed in parallel with one of these setups, at least three jobs of J_1 with total processing time at least $2y$ are required. Furthermore, since $d_{3r+1} = 2y+2$, job $3r+1$ must be setup by time $2y+1$. Thus, in the interval $[0, 2y+1]$, there is one unit of unavoidable machine idle time, three setups and $2y$ units of processing for jobs in J_1, the setup of $2y-3$ type 1 jobs in J_3, and the setup of job $3r+1$, the total time for which is $1+(3+2y)+(2y-3)+1=4y+2$. Thus, no further setup or processing operations are possible in the interval $[0, 2y+1]$, so job $3r+1$ must be processed in the interval $[2y+1, 2y+2]$. Since the server is always busy, there is also a setup in $[2y+1, 2y+2]$. It follows that the processing of the three jobs in $\{1, \ldots, 3r\}$ which are processed in $[0, 2y+1]$ does not continue after time $2y+1$. Thus, the processing times of these three jobs sum to exactly $2y$. A similar argument for each subsequent time interval of length $2y+1$ shows that $L_{\max} \leqslant 0$ implies the existence of a solution to 3-Partition. □

We now consider problems with unit processing times. The EDD sequence in which jobs appear in nondecreasing order of their due dates is useful in this analysis.

Theorem 5.2. *Problem* $P, S|p_j = 1|L_{\max}$ *can be solved optimally in* $O(n \log n)$ *time, by using List, with* σ *denoting an EDD sequence of the jobs.*

Proof. Theorem 3.2 shows that problem $P, S|p_j = 1|L_{\max}$ is equivalent to problem $1||L_{\max}$ with $n' = n$, and $p'_j = s_j$ and $d'_j = d_j - 1$, for $j = 1, \ldots, n$. Jackson [16] shows that an optimal solution of problem $1||L_{\max}$ is obtained in $O(n' \log n')$ time by sequencing the jobs in EDD order. □

6. Total completion time problems

We begin by considering problems with arbitrary processing times. For problem $P2, S|s_j = 1|\sum_{j=1}^{n} C_j$, we propose an algorithm, **SPT**, which is simply **List** with σ denoting an SPT sequence in which jobs appear in nondecreasing order of their processing times.

Theorem 6.1. *Algorithm* **SPT** *solves problem* $P2, S|s_j = 1|\sum_{j=1}^{n} C_j$ *optimally in* $O(n \log n)$ *time.*

Proof. We note that, in any schedule, there is unavoidable idle time in the interval $[0, 1]$ on one of the machines. We therefore assume, without loss of generality, that machine M_2 becomes available at time 1. From the SPT ordering, the schedule on machine M_1 is longer than that on machine M_2 immediately after the scheduling of

job $\sigma(1)$ by Algorithm **SPT**. Similarly, the schedule on M_2 is longer than that on M_1 immediately after job $\sigma(2)$ has been scheduled. The same argument can be applied to each machine alternately, i.e., after job $\sigma(3)$ on M_1, after job $\sigma(4)$ on M_2, and so on. It follows that the completion times of any two jobs scheduled by **SPT** do not coincide, and consequently no server interference (when two or more machines simultaneously need the server) occurs. Thus, machine idle time never occurs after time 1. Consequently, the total completion time of the solution delivered by **SPT** is the same as in an instance of the classical scheduling problem $P2||\sum_{j=1}^{n'} C_j$ with $n'=n$ and processing times defined by $p'_j = p_j + 1$ for $j=1,\ldots,n$, where machine M_2 again becomes available at time 1. The latter problem is a relaxation of the problem being considered, since server interference is not considered there. Since **SPT** generates an optimal schedule for $P2||\sum_{j=1}^{n'} C_j$ [3], it also generates an optimal schedule for problem $P2,\ S|s_j=1|\sum_{j=1}^{n} C_j$. The dominant time requirement is for ordering the jobs by SPT, which takes $O(n\log n)$ time. □

Algorithm **SPT** does not generalize to give an optimal solution for problem $Pm,\ S|s_j=1|\sum_{j=1}^{n} C_j$, for $m>2$. The complexity of this problem remains open.

We next consider a more general problem.

Theorem 6.2. *The recognition version of problem* $P2,\ S|s_j = s|\sum_{j=1}^{n} C_j$ *is unary* NP-*complete.*

Proof. We use a reduction from Numerical Matching with Target Sums, as defined in the proof of Theorem 4.2. Given an arbitrary instance of Numerical Matching with Target Sums, consider the following instance of $P2,\ S|s_j = s|\sum_{j=1}^{n} C_j$:

$$
\begin{aligned}
&n = 2K^3 + 3r,\\
&s_j = K^2, \quad p_j = K + x_j, \quad j = 1,\ldots,r,\\
&s_j = K^2, \quad p_j = 2K + y_{j-r}, \quad j = r+1,\ldots,2r,\\
&s_j = K^2, \quad p_j = 3K^2 + 3K + z_{j-2r}, \quad j = 2r+1,\ldots,3r,\\
&s_j = K^2, \quad p_j = K^6 - K^2, \quad j = 3r+1,\ldots,3r+2K^3,\\
&C = C' + C'',
\end{aligned}
$$

where $K = \sum_{j=1}^{r}(x_j + y_j + z_j) + 20r^2$, $b = \sum_{j=1}^{r}(x_j + y_j + z_j)$, $C' = r/2[(4K^2 + 3K + b) + (5K^2 + 4K + b) + (6K^2 + 6K + b)] + 3(6K^2 + 6K)[1 + \cdots + (r/2 - 1)] + r/2[(2K^2 + K + b) + (3K^2 + 3K + b) + (7K^2 + 6K + b)] + 3(6K^2 + 6K)[1 + \cdots + (r/2 - 1)]$, and $C'' = K^9(K^3 + 1) + K^3[(r/2)(6K^2 + 6K) + b/2] + K^3[(r/2)(6K^2 + 6K) + b/2 + K^2]$.

We now show that there exists a schedule for the constructed instance of $P2,\ S|s_j = s|\sum_{j=1}^{n} C_j$ with $\sum_{j=1}^{n} C_j \leqslant C$ if and only if there exists a solution to Numerical Matching with Target Sums.

($\Rightarrow$) Note that jobs $1,\ldots,3r$ are identical to those used in the proof of Theorem 4.2. Therefore, consider a schedule similar to that shown in Fig. 1, where jobs $3r+1,\ldots,3r+2K^3$ follow the others, with K^3 of them being processed on each machine. Further, no idle time will occur following the completion of jobs $1,\ldots,3r$. To obtain

an upper bound on $\sum_{j=1}^{n} C_j$, we replace the contribution to C_j that involves a sum of the elements of Numerical Matching with Target Sums, for $j \in \{1,\ldots,3r\}$, by its upper bound b. The setups for jobs of $\{3r+1,\ldots,3r+2K^2\}$ start on M_1 and M_2 at times $(r/2)(6K^2+6K)+b/2+K^2$ and $(r/2)(6K^2+6K)+b/2$, respectively. A simple computation therefore shows that $\sum_{j=1}^{n} C_j \leqslant C = C' + C''$, where C' and C'' represent the contributions of jobs $1,\ldots,3r$ and $3r+1,\ldots,3r+2K^2$, respectively.

($\Leftarrow$) First, we show that, in any schedule with $\sum_{j=1}^{n} C_j \leqslant C$, jobs $1,\ldots,3r$ are each completed before the long jobs $3r+1,\ldots,3r+2K^3$. It is useful for this analysis to derive upper bounds on C' and C''. By noting that $b < K$, we obtain

$$C' < 9rK^2(1+r)/2 + rK(9r+11)/2 < 9rK^2(1+r)/2 + K^2 < K^3, \tag{1}$$

and also using $K < K^2$ we deduce that

$$C'' < K^9(K^3+1) + K^3(12rK^2+2K^2) < K^9(K^3+1) + K^6 - K^3. \tag{2}$$

Consider any schedule in which one or more of the jobs $1,\ldots,3r$ is completed after a long job and therefore has a completion time of at least K^6. To obtain a lower bound on the total completion time of the long jobs, we schedule K^3 jobs on each machine with completion times at least $K^6, 2K^6, \ldots, K^9$. Thus, the total completion time of this schedule is bounded by

$$\sum_{j=1}^{n} C_j > K^9(K^3+1) + K^6 > C,$$

where the last inequality is obtained from (1) and (2). It follows that, on each machine, no long job is scheduled before any of the jobs $1,\ldots,3r$.

We now show that, apart from during the interval $[0,K^2]$, there is no machine idle time before the long jobs are processed. We derive a lower bound on the total completion time of an optimal schedule by using a relaxation which allow setups on both machines to be performed simultaneously. Algorithm **SPT** generates an optimal schedule for this relaxed problem in which no long job is scheduled to start before one of the jobs $1,\ldots,3r$. Note that half of the long jobs are scheduled on each machine. If processing of the long jobs starts on M_1 and M_2 at times t_1 and t_2, respectively, then a lower bound from the completion times of the long jobs is

$$\sum_{j=1}^{n} C_j \geqslant K^9(K^3+1) + K^3(t_1+t_2),$$

from which we use (1) and the definition of C'' to deduce that

$$\sum_{j=1}^{n} C_j - C > K^3(t_1+t_2 - r(6K^2+6K) - b - K^2 - 1). \tag{3}$$

Since the total setup and processing time of jobs $1,\ldots,3r$ is $r(6K^2+6K)+b$, and the unavoidable machine idle time is K^2, we have $t_1+t_2 \geqslant r(6K^2+6K)+b+K^2$. Moreover, if there is inserted idle time, then since t_1+t_2 is integer, $t_1+t_2 \geqslant r(6K^2+6K)+b+K^2+1$, and thus from (3) we have $\sum_{j=1}^{n} C_j > C$. Therefore, in any schedule

with $\sum_{j=1}^{n} C_j \leqslant C$, jobs $1,\ldots,3r$ are scheduled without machine idle time (except in the interval $[0,K^2]$) and none is preceded by a long job on the same machine.

We now show that the schedule for jobs $1,\ldots,3r$ is of the form shown in Fig. 1. Consider the allocation of jobs $1,\ldots,3r$ between machines M_1 and M_2. Provided that there is no machine idle time, the total completion time of the long jobs is equal to C'', and is not affected by this allocation. Any schedule that starts with some job j, for $j \in \{1,\ldots,2r\}$, will have machine idle time because the server is required throughout the interval $[K^2,2K^2]$ for a setup on the other machine. Thus, all jobs are completed after time $2K^2$. Also, the completion of two jobs within an interval $[hK^2,(h+1)K^2)$, for $h=2,\ldots,3r+1$, causes idle time because of the simultaneous requirement for the server to setup the two following jobs. Thus, $2K^2,\ldots,(3r+1)K^2$ are lower bounds on the completion times of the first $3r$ jobs. If there is some interval $[hK^2,(h+1)K^2)$, where $h \in \{2,\ldots,3r+1\}$, within which no job is completed, then lower bounds on the completion times of the first $3r$ jobs are $2K^2,\ldots,3rK^2,(3r+2)K^2$. Thus, $\sum_{j=1}^{n} C_j - C'' \geqslant 9rK^2(1+r)/2 + K^2 > C'$ using (1). Therefore, in a schedule with $\sum_{j=1}^{n} C_j \leqslant C$, one of the jobs $1,\ldots,3r$ is completed in each of the intervals $[hK^2,(h+1)K^2)$, for $h=2,\ldots,3r+1$.

As observed above, neither any of the jobs $1,\ldots,2r$, nor a long job, can be processed first. Thus, some job $k \in \{2r+1,\ldots,3r\}$ is processed first, for example on M_2. Then the first two jobs processed on M_1 are i and j, where each is chosen from the subset $\{1,\ldots,2r\}$, since completion of jobs in each of the intervals $[2K^2,3K^2)$ and $[3K^2,4K^2)$ is impossible for any other choice. Moreover, the third job processed on M_1 is from the subset $\{2r+1,\ldots,3r\}$, since otherwise one of the jobs $1,\ldots,2r$ would be processed and two jobs would complete in the interval $[4K^2,5K^2)$, causing server interference and thus idle time. A similar argument is used for the other groups of three jobs. Thus, while each job from $\{2r+1,\ldots,3r\}$ is processed, two jobs from $\{1,\ldots,2r\}$ are setup and processed, and there is an additional setup for a later job.

Finally, we show that $p_i+p_j=3K+z_{k-2r}$. If $p_i+p_j > 3K+z_{k-2r}$, then machine idle time results from the simultaneous requirement on both machines for the server at time $4K^2+3K+z_{k-2r}$. On the other hand, if $p_i+p_j < 3K+z_{k-2r}$, then from the definition of Numerical Matching with Target Sums there exist some subsequent jobs i', j' and k', which are defined analogously to i, j and k, for which $p_{i'}+p_{j'} > 3K+z_{k'}$, and machine idle time results. Therefore $p_i+p_j=3K+z_{k-2r}$, from which we deduce that $i \in \{1,\ldots,r\}$ and $j \in \{r+1,\ldots,2r\}$, or vice versa. Thus, $x_i+y_{j-r}=z_{k-2r}$. Applying the same argument to each subsequent group of three jobs, we obtain a solution to Numerical Matching with Target Sums. □

We now consider unit processing time problems with $\sum_{j=1}^{n} C_j$ objective.

Theorem 6.3. *Problem* $P,\ S|p_j=1,\ s_j=s|\sum_{j=1}^{n} C_j$ *can be solved optimally in constant time by scheduling the setup and processing for job* j *in the interval* $[(j-1)s, js+1]$ *on machine* M_1 *if* j *is odd, or on* M_2 *if* j *is even, for* $j=1,\ldots,n$.

Proof. Theorem 3.1 shows that problem $P,\ S|p_j=1,\ s_j=s|\sum_{j=1}^{n} C_j$ is equivalent to problem $1|p'_j=p'|\sum_{j=1}^{n'} C_j$ with $n'=n$ and $p'_j=p'=s$ for $j=1,\ldots,n$. Since all jobs are identical, an optimal solution for problem $1|p'_j=p'|\sum_{j=1}^{n'} C_j$ schedules job j in the interval $[(j-1)p',jp']$, which provides the desired schedule for problem $P,\ S|p_j=1,\ s_j=s|\sum_{j=1}^{n} C_j$. □

Theorem 6.4. *Problem* $P,\ S|p_j=1|\sum_{j=1}^{n} C_j$ *can be solved optimally in* $O(n\log n)$ *time by using* List, *with* σ *denoting a nondecreasing setup time ordering of the jobs.*

Proof. Theorem 3.1 shows that problem $P,\ S|p_j=1|\sum_{j=1}^{n} C_j$ is equivalent to problem $1||\sum_{j=1}^{n'} C_j$ with $n'=n$, and $p'_j=s_j$ for $j=1,\ldots,n$. Smith [33] shows that an optimal solution of problem $1||\sum_{j=1}^{n'} C_j$ is obtained in $O(n'\log n')$ time by sequencing the jobs in nondecreasing processing time order. □

Turning now to problems with weighted sum of completion times objective, or $\sum_{j=1}^{n} w_jC_j$, we begin with a negative result.

Theorem 6.5. *The recognition version of problem* $P2,\ S|s_j=1|\sum_{j=1}^{n} w_jC_j$ *is binary* NP-*complete.*

Proof. Consider the following classical scheduling problem which is known to be binary NP-complete.

$P2||\sum_{j=1}^{n'} w'_jC_j$ [26]: given n' jobs, each job j having processing time p'_j and positive weight w'_j, and an integer C', does there exist a nonpreemptive two machine schedule for which $\sum_{j=1}^{n'} w'_jC_j\leqslant C'$?

Given an arbitrary instance of $P2||\sum_{j=1}^{n'} w'_jC_j$, consider the following instance of $P2,\ S|s_j=1|\sum_{j=1}^{n} w_jC_j$:

$$
\begin{aligned}
&n=n',\\
&p_j=n'\left(\sum_{i=1}^{n'} w'_i\right)p'_j-1, \qquad j=1,\ldots,n,\\
&w_j=w'_j, \qquad\qquad\qquad\quad j=1,\ldots,n,\\
&C=n'(C'+1)\sum_{j=1}^{n'} w'_j-1.
\end{aligned}
$$

We now show that there exists a schedule for this instance of $P2,\ S|s_j=1|\sum_{j=1}^{n} w_jC_j$ with $\sum_{j=1}^{n} w_jC_j\leqslant C$ if and only if there exists a schedule for the instance of $P2||\sum_{j=1}^{n'} w'_jC_j$ with $\sum_{j=1}^{n'} w'_jC_j\leqslant C'$.

($\Rightarrow$) Any schedule for $P2||\sum_{j=1}^{n'} w_jC_j$ with $\sum_{j=1}^{n'} w_jC_j\leqslant C'$ can be used to create a schedule for $P2,\ S|s_j=1|\sum_{j=1}^{n} w_jC_j$ by first multiplying all processing and completion times by $n'\sum_{j=1}^{n'} w'_j$ and then changing the first unit of processing time into a setup

time. When there are simultaneous requests for the server, one unit of idle time is inserted on one of the machines. Since at most $n'-1$ units of idle time are inserted, we obtain $\sum_{j=1}^{n} w_j C_j \leqslant n'C' \sum_{j=1}^{n'} w'_j + (n'-1)\sum_{j=1}^{n} w'_j \leqslant n'(C'+1)\sum_{j=1}^{n'} w'_j - 1 = C$.

($\Leftarrow$) Consider a schedule for $P2,\ S|s_j=1|\sum_{j=1}^{n} w_j C_j$ with $\sum_{j=1}^{n} w_j C_j \leqslant C$. To construct a schedule for $P2||\sum_{j=1}^{n'} w_j C_j$, first regard each of the setup times as part of the processing time, and remove all idle time from the schedule. Then divide all processing and completion times by $n'\sum_{j=1}^{n'} w'_j$. Since by assumption p'_j is integer for $j=1,\ldots,n'$, this yields a schedule for $P2||\sum_{j=1}^{n'} w'_j C_j$ with $\sum_{j=1}^{n'} w'_j C_j \leqslant \lfloor C/(n'\sum_{j=1}^{n'} w'_j)\rfloor = \lfloor (C'+1) - 1/n'\sum_{j=1}^{n'} w'_j\rfloor = C'$. □

This problem remains open as to unary NP-completeness. However, the assumption of unit processing times makes problems with this objective much more tractable.

Theorem 6.6. *Problem* $P,\ S|p_j=1|\sum_{j=1}^{n} w_j C_j$ *can be solved optimally in* $O(n\log n)$ *time by using* List, *with* σ *denoting a nondecreasing* s_j/w_j *ordering of the jobs.*

Proof. Theorem 3.1 shows that problem $P,\ S|p_j=1|\sum_{j=1}^{n} w_j C_j$ is equivalent to problem $1||\sum_{j=1}^{n'} w'_j C_j$ with $n'=n$, and $p'_j=s_j$ and $w'_j=w_j$, for $j=1,\ldots,n$. Smith [33] shows that an optimal solution of problem $1||\sum_{j=1}^{n'} w'_j C_j$ is obtained in $O(n'\log n')$ time by sequencing the jobs in nondecreasing p'_j/w'_j order. □

7. Number of late jobs problems

We begin with a negative result.

Corollary 7.1. *The recognition version of problem* $P2,\ S|s_j=1|\sum_{j=1}^{n} U_j$ *is unary* NP-*complete.*

Proof. The recognition versions of problems $P2,\ S|s_j=1|\sum_{j=1}^{n} U_j$ and $P2,\ S|s_j=1|L_{\max}$ with threshold costs of zero in both cases are equivalent. Since Theorem 5.1 shows that this recognition version of problem $P2,\ S|s_j=1|L_{\max}$ is unary NP-complete, it follows that this recognition version of problem $P2,\ S|s_j=1|\sum_{j=1}^{n} U_j$ is also unary NP-complete. □

For problems of minimizing the (weighted) number of late jobs with unit processing times, we deduce polynomial time algorithms and NP-completeness results from Theorem 3.2. Specifically, by setting $n'=n$, $d'_j=d_j-1$ and $w'_j=w_j$ for $j=1,\ldots,n$, we obtain that problem $P,\ S|p_j=1|\sum(w_j)U_j$ is equivalent to problem $1||\sum(w'_j)U_j$ with $p'_j=s_j$ for $j=1,\ldots,n$, and that problem $P,\ S|p_j=1,\ s_j=s|\sum(w_j)U_j$ is equivalent to problem $1|p'_j=p'|\sum(w'_j)U_j$ with $p'_j=p'=s$ for $j=1,\ldots,n$. The complexity results which follow from this discussion appear in Table 2.

Table 2
Equivalence results for number of late jobs problems

Common server problem	Equivalent classical problem	Reference to classical result	Complexity of classical problem	Complexity of common server problem
$P,S\|p_j=1,s_j=s\|\sum_{j=1}^{n}U_j$	$1\|p'_j=p'\|\sum_{j=1}^{n'}U_j$	Monma [29]	$O(n')$	$O(n)$
$P,S\|p_j=1\|\sum_{j=1}^{n}U_j$	$1\|\|\sum_{j=1}^{n'}U_j$	Moore [30]	$O(n'\log n')$	$O(n\log n)$
$P,S\|p_j=1,s_j=s\|\sum_{j=1}^{n}w_jU_j$	$1\|p'_j=p'\|\sum_{j=1}^{n'}w'_jU_j$	Lawler [23]	$O(n'\log n')$	$O(n\log n)$
$P,S\|p_j=1\|\sum_{j=1}^{n}w_jU_j$	$1\|\|\sum_{j=1}^{n'}w'_jU_j$	Karp [18]	Binary NPC	Binary NPC
		Lawler and Moore [25]	$O(n'\sum_{j=1}^{n'}p'_j)$	$O(n\sum_{j=1}^{n}s_j)$

Table 3
Equivalence results for total tardiness problems

Common server problem	Equivalent classical problem	Reference to classical result	Complexity of classical problem	Complexity of common server problem
$P,S\|p_j=1,s_j=s\|\sum_{j=1}^{n}T_j$	$1\|p'_j=p'\|\sum_{j=1}^{n'}T_j$	Emmons [5]	$O(n'\log n')$	$O(n\log n)$
$P,S\|p_j=1\|\sum_{j=1}^{n}T_j$	$1\|\|\sum_{j=1}^{n'}T_j$	Du and Leung [4]	Binary NPC	Binary NPC
		Lawler [24]	$O(n'^4\sum_{j=1}^{n'}p'_j)$	$O(n^4\sum_{j=1}^{n}s_j)$
$P,S\|p_j=1,s_j=s\|\sum_{j=1}^{n}w_jT_j$	$1\|p'_j=p'\|\sum_{j=1}^{n'}w'_jT_j$	Lawler [22]	$O(n'^3)$	$O(n^3)$
$P,S\|p_j=1\|\sum_{j=1}^{n}w_jT_j$	$1\|\|\sum_{j=1}^{n'}w'_jT_j$	Lawler [24]	Unary NPC	Unary NPC

8. Total tardiness problems

We begin with a negative result.

Corollary 8.1. *The recognition version of problem* $P2,\ S|s_j=1|\sum_{j=1}^{n}T_j$ *is unary* NP-*complete.*

Proof. The recognition versions of problems $P2,\ S|s_j=1|\sum_{j=1}^{n}T_j$ and $P2,\ S|s_j=1|L_{\max}$ with threshold costs of zero in both cases are equivalent. Since Theorem 5.1 shows that this recognition version of problem $P2,\ S|s_j=1|L_{\max}$ is unary NP-complete, it follows that this recognition version of problem $P2,\ S|s_j=1|\sum_{j=1}^{n}T_j$ is also unary NP-complete. □

For the problems of minimizing the total (weighted) tardiness with unit processing times, we use Theorem 3.2 in the same way as in Section 7. This analysis provides the complexity results that appear in Table 3.

9. Concluding remarks

In this paper, we describe a number of results which map the computational complexity of scheduling problems in a parallel machine environment with a common server. The type of server will vary among production settings to which our models can be applied. The server could, for example, be a human operator or a piece of equipment used to initialize jobs.

One important application of these scheduling models occurs in modern manufacturing systems, where the common server is a robot. With the increasing use of automated material handling systems in manufacturing, it seems certain that robots will continue to play an important part in the design of efficient scheduling systems. Related works by Hall et al. [10,11], Kamoun et al. [17] and Sriskandarajah et al. [34] resolve a number of algorithmic, computational complexity, and practical, issues arising in robotic cells. Hall and Sriskandarajah [12] review related scheduling issues in no-wait and bufferless environments. In the present paper, the server performs setups in a classical parallel machine shop.

In subsequent work, Kravchenko and Werner [21] describe an $O(n\sum_{j=1}^{n} p_j)$ time algorithm for problem $P2,\ S|s_j = 1|C_{\max}$. Thus, the only remaining problems with completely open complexity in the environment discussed here are $P,\ S|\ |\sum_{j=1}^{n} C_j$ and its special cases with $s_j = 1$ and $s_j = s$. In addition, problem $P2,\ S|s_j = 1|\sum_{j=1}^{n} w_j C_j$ is known to be binary NP-complete in its recognition form, but is open with respect to pseudo-polynomial time solvability. There are also other topics worthy of consideration. These include the computational complexity of scheduling problems with a common server in flowshop, jobshop and openshop environments. The flowshop environment differs from the robotic cell mentioned above, in that intermediate buffers may exist between the machines in a flowshop. In each of these environments, more general problems involving the use of two or more servers may also be studied.

Acknowledgements

This research was supported in part by NATO Collaborative Research Grant CRG 950773, by the Summer Fellowship Program (Fisher College of Business, The Ohio State University), by INTAS (Projects INTAS-93-257 and INTAS-93-257-Ext), and by Natural Sciences and Engineering Research Council of Canada (Grant no. OGP0104900). An anonymous referee provided helpful comments on an earlier draft of this paper.

References

[1] J.E. Aronson, Two heuristics for the deterministic, single operator, multiple machine, multiple run cyclic scheduling problem, J. Oper. Management 4 (1984) 159–173.
[2] K.R. Baker, Introduction to Sequencing and Scheduling, Wiley, New York, 1974.
[3] R.W. Conway, W.L. Maxwell, L.W. Miller, Theory of Scheduling, Addison-Wesley, Reading, MA, 1967.

[4] J. Du, J.Y.-T. Leung, Minimizing total tardiness on one machine is NP-hard, Math. Oper. Res. 15 (1990) 483–495.
[5] H. Emmons, One-machine sequencing to minimize certain functions of job tardiness, Oper. Res. 17 (1969) 701–715.
[6] T. Ganesharajah, N.G. Hall, C. Sriskandarajah, Design and operational issues in AGV-served manufacturing systems, Ann. Oper. Res. 76 (1998) 109–154.
[7] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman & Company, San Francisco, 1979.
[8] C.A. Glass, Y.M. Shafransky, V.A. Strusevich, Scheduling for parallel dedicated machines with a single server, Working Paper OR86, Faculty of Mathematical Studies, University of Southampton, UK, 1996.
[9] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic machine scheduling: a survey, Ann. Discrete Math. 5 (1979) 287–326.
[10] N.G. Hall, H. Kamoun, C. Sriskandarajah, Scheduling in robotic cells: classification, two and three machine cells, Oper. Res. 45 (1997) 421–439.
[11] N.G. Hall, H. Kamoun, C. Sriskandarajah, Scheduling in robotic cells: complexity and steady state analysis, European J. Oper. Res. 109 (1998) 43–65.
[12] N.G. Hall, C. Sriskandarajah, A survey of machine scheduling problems with blocking and no-wait in process, Oper. Res. 44 (1996) 510–525.
[13] J. Hartley, Robots at Work, North-Holland, Amsterdam, 1983.
[14] G.M. Hull, Case study of the design and installation of a robot-loaded machining cell, Proceedings of the Fourth International Conference on Flexible Manufacturing Systems, 1985.
[15] H. Inaba, S. Sakakibara, Flexible automation — unmanned machining and assembly cells with robots, Proceedings of the First International Conference on Flexible Manufacturing Systems, 1982.
[16] J.R. Jackson, Scheduling a production line to minimize maximum tardiness, Management Science Research Project, University of California, Los Angeles, CA, 1955.
[17] H. Kamoun, N.G. Hall, C. Sriskandarajah, Scheduling in robotic cells: heuristics and cell design, Oper. Res., 1999, to appear.
[18] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller, J.W. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 85–103.
[19] C.P. Koulamas, Scheduling two parallel semiautomatic machines to minimize machine interference, Comput. Oper. Res. 23 (1996) 945–956.
[20] C.P. Koulamas, M.L. Smith, Look-ahead scheduling for minimizing machine interference, Internat. J. Production Res. 26 (1988) 1523–1533.
[21] S.A. Kravchenko, F. Werner, Parallel machine scheduling problems with a single server, Math. Comput. Modelling 26 (1997) 1–11.
[22] E.L. Lawler, On scheduling problems with deferral costs, Management Sci. 11 (1964) 280–288.
[23] E.L. Lawler, Sequencing to minimize the weighted number of tardy jobs, RAIRO Recherche Opérationnelle S10 (5) (1976) 27–33.
[24] E.L. Lawler, A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness, Ann. Discrete Math. 1 (1977) 331–342.
[25] E.L. Lawler, J.M. Moore, A functional equation and its application to resource allocation and sequencing problems, Management Sci. 16 (1969) 77–84.
[26] J.K. Lenstra, A.H.G. Rinnooy Kan, P. Brucker, Complexity of machine scheduling problems, Annals of Discrete Mathematics 1 (1977) 343–362.
[27] J.G. Miller, W.L. Berry, Heuristic methods for assigning men to machines: an experimental analysis, AIIE Trans. 6 (1974) 97–104.
[28] J.G. Miller, W.L. Berry, The assignment of men to machines: an application of branch and bound, Decision Sci. 8 (1977) 56–72.
[29] C.L. Monma, Linear-time algorithms for scheduling on parallel processors, Oper. Res. 30 (1982) 116–124.
[30] J.M. Moore, An n job, one machine sequencing algorithm for minimizing the number of late jobs, Management Sci. 15 (1968) 102–109.
[31] C.H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, NJ, 1982.
[32] S.P. Sethi, C. Sriskandarajah, G. Sorger, J. Blazewicz, W. Kubiak, Sequencing of parts and robot moves in a robotic cell, Internat. J. Flexible Manufacturing Systems 4 (1992) 331–358.

[33] W.E. Smith, Various optimizers for single-stage production, Naval Res. Logistics Quart. 3 (1956) 59–66.
[34] C. Sriskandarajah, N.G. Hall, H. Kamoun, Scheduling large robotic cells without buffers, Ann. Oper. Res. 76 (1998) 287–321.
[35] K.E. Stecke, J.E. Aronson, Review of operator/machine interference models, Internat. J. Production Res. 23 (1985) 129–151.
[36] W.E. Wilhelm, S.C. Sarin, A structure for sequencing robot activities in machine loading applications, Internat. J. Production Res. 23 (1985) 47–64.