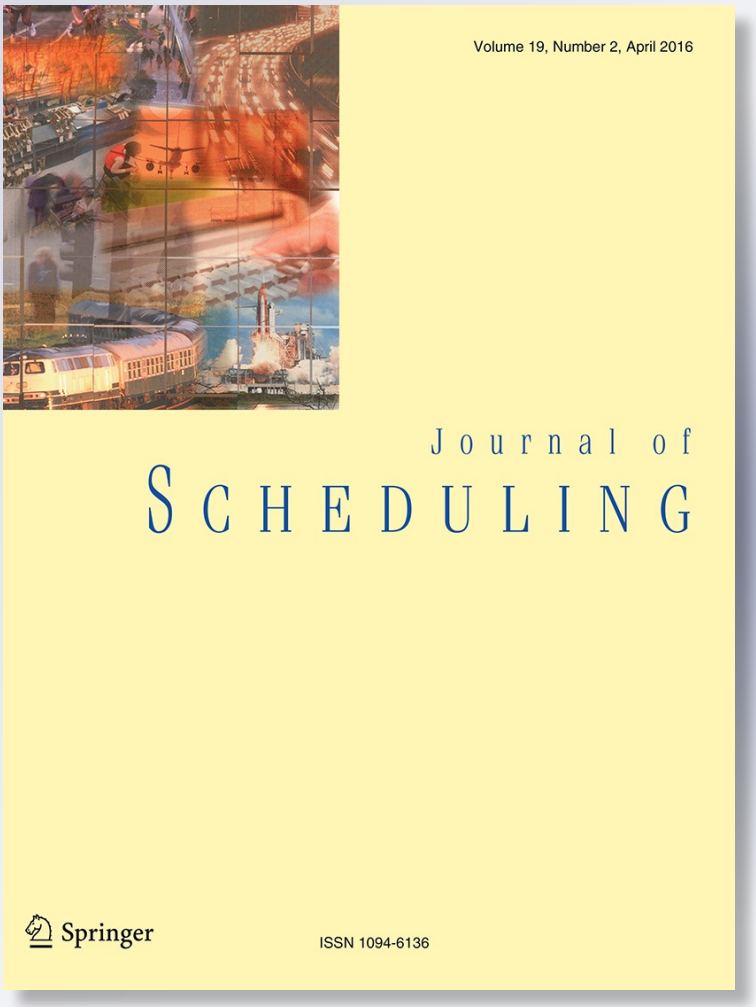
Relocation scheduling subject to fixed processing sequences

Bertrand M. T. Lin, F. J. Hwang & Alexander V. Kononov

Journal of Scheduling

ISSN 1094-6136
Volume 19
Number 2

J Sched (2016) 19:153-163
DOI 10.1007/s10951-015-0455-8



Springer

J Sched (2016) 19:153–163
DOI 10.1007/s10951-015-0455-8

Relocation scheduling subject to fixed processing sequences

Bertrand M. T. Lin[1] · **F. J. Hwang**[2] · **Alexander V. Kononov**[3]

Published online: 28 September 2015

Abstract This study addresses a relocation scheduling problem that corresponds to resource-constrained scheduling on two parallel dedicated machines where the processing sequences of jobs assigned to the machines are given and fixed. Subject to the resource constraints, the problem is to determine the starting times of all jobs for each of the six considered regular performance measures, namely, the makespan, total weighted completion time, maximum lateness, total weighted tardiness, weighted number of tardy jobs, and number of tardy jobs. By virtue of the proposed dynamic programming framework, the studied problem for the minimization of makespan, total weighted completion time, or maximum lateness can be solved in $O(n_1n_2(n_1+n_2))$ time, where n_1 and n_2 are the numbers of jobs on the two machines. The simplified case with a common job processing time can be solved in $O(n_1n_2)$ time. For the objective function of total weighted tardiness or weighted number of tardy jobs, this problem is proved to be NP-hard in the ordinary sense, and the case with a common job processing length is solvable in $O(n_1n_2\min\{n_1,n_2\})$ time. The studied problem for the minimization of number of tardy jobs is solvable in $O(n_1^2n_2^2(n_1+n_2)^2)$ time. The solvability of the common-processing-time problems can be generalized to the m-machine cases, where $m \geq 3$.

Keywords Relocation problem · Resource-constrained scheduling · Parallel dedicated machines · Fixed sequence · Dynamic programming · NP-hardness

✉ Bertrand M. T. Lin
bmtlin@mail.nctu.edu.tw

F. J. Hwang
feng-jang.hwang@uts.edu.au

Alexander V. Kononov
alvenko@math.nsc.ru

[1] Department of Information Management and Finance, Institute of Information Management, National Chiao Tung University, Hsinchu, Taiwan

[2] School of Mathematical and Physical Sciences, University of Technology Sydney, Ultimo, Australia

[3] Sobolev Institute of Mathematics, Novosibirsk, Russia

1 Introduction

Resource-constrained scheduling has been receiving considerable research attention for decades. To deal with the complicated decision-making processes, solution approaches usually consist of three major phases, namely, (1) assigning jobs to machines, (2) sequencing the jobs on each machine, and (3) determining the starting time of each job on each machine in compliance with resource availability. Although processing capabilities of machines could be regarded as limited resources in scheduling decisions, resource constraints addressed in this paper refer to the limited availability of other types of resources, such as capital, housing capacity, or inventory. The research setting of this study corresponds to the timing phase and highlights the observation that the decision of the last phase itself could be a challenge.

This paper investigates a variant of the relocation problem that is formulated as resource-constrained scheduling from a public housing redevelopment project in Boston (Kaplan 1986; Kaplan and Berman 1988). The resource constraints considered in relocation scheduling are different from the conventional ones in project scheduling in the sense that the amount of resource released by a completed job can be larger than, equal to, or smaller than that the job has acquired for commencing its processing. In the basic relocation problem, there are several buildings to be redeveloped. The municipal authority provides a budget of temporary housing units

for accommodating the tenants during the redevelopment process. Any building can be demolished only if its tenants are evacuated and there are sufficient temporary housing units for the tenants. The erected building will have a new capacity, which can be used for housing the tenants of other remaining buildings. The new capacity of the building is not necessarily the same as the original capacity. No preemption is allowed and the construction team can process a building at a time. A schedule is *feasible* if all buildings can be successfully processed in the sense that all tenants are housed at any time. If we interpret a building, its original capacity and new capacity as a job, its resource requirement and resource yield, respectively, then relocation scheduling becomes a resource-constrained scheduling problem. Kaplan and Amir (1988) show that determining the minimum initial resource level that guarantees the existence of feasible schedules is equivalent to minimizing the makespan in two-machine flow shop scheduling (Johnson 1954).

In the first setting of the relocation problem, job processing times and multiple working crews are considered, and the objective is to construct a feasible schedule with the minimum makespan (Kaplan 1986). While Amir and Kaplan (1988) prove the problem with two identical working crews to be NP-hard, whether this problem is strongly NP-hard or not had remained open until Kononov and Lin (2006) give a proof for a restricted case where two parallel dedicated machines with unit-execution-time jobs are considered, and the amount of the resource returned by any job is greater than the amount of the resource this job requires for its processing. In the setting of the two parallel dedicated machines, the jobs are beforehand divided into two subsets which are dispatched to the respective machines. Each job must be processed on its dedicated machine, and the two machines can work in parallel if the resource is sufficient at any moment. For the objective function of total completion time, Kononov and Lin (2010) show that the single-machine relocation problem is strongly NP-hard even if all jobs make positive contributions. These complexity results together imply that both makespan minimization and total completion time minimization in the relocation problem on two parallel dedicated machines are strongly NP-hard. Additionally, it follows that the studied problems with due-date-related performance metrics, such as maximum lateness, total tardiness, and number of tardy jobs, are also strongly NP-hard. In this study, we address the relocation problem on two parallel dedicated machines subject to fixed processing sequences, the assumption of which is mainly motivated by the perspective of decision decomposition.

Given an intractable problem involving multiple interrelated decision-making phases, investigating a specific phase obtained by simplifying the decision processes could yield essential insights into the underlying problem. The developed solution procedure for the sub-problem can be used as the building block of that for the original problem. Due to the computational intractability of the relocation problem on parallel machines, different solution approaches, including branch-and-bound algorithms and search-based or population-based meta-heuristics, can be deployed to tackle the problem. For NP-hard problems, most of the solution algorithms are developed with sequence-based representations, which can be regarded as assignments of jobs to machines' positions. If the problem with fixed sequences on dedicated machines can be solved in polynomial time, then efficient procedures for computing the optimal objective values of given (partial) processing sequences, by determining the starting times of the reconstruction of all buildings, are crucial to the performance of the developed solution approaches, either exact or approximation. On the other hand, if the fixed-sequence problem is proved to be NP-hard, then solution representation schemes other than the sequence-based one would be required. This stimulates the study with the assumption that the processing sequences on the dedicated machines are given and fixed a priori. Although determining optimal schedules from given processing sequences of jobs/operations is easy in most scheduling problems, we will learn from the results shown in the following sections that it is not trivial for the studied problem and even NP-hard for some objective functions. Other justifications of the assumption of fixed job sequences can also be found in several previous studies (Cheng et al. 2000; Hwang et al. 2012, 2014; Hwang and Lin 2011; Lin and Cheng 2006; Lin and Hwang 2011; Ng and Kovalyov 2007; Shafransky and Strusevich 1998).

This paper adopts the notation RPD used in Kononov and Lin (2006) to denote the relocation problem on parallel dedicated machines, and RPDm signifies the case of m machines. This study investigates six regular performance measures, including the makespan, total weighted completion time, maximum lateness, total weighted tardiness, weighted number of tardy jobs, and number of tardy jobs, i.e., the objective function $\gamma \in \{C_{\max}, \sum w_j C_j, L_{\max}, \sum w_j T_j, \sum w_j U_j, \sum U_j\}$. Section 2 gives formal statements of the problem definitions and preliminary properties. In Sect. 3, we propose a backward dynamic program for the objective function of makespan, total weighted completion time, or maximum lateness. Three due-date-related criteria, including the total weighted tardiness, the weighted number of tardy jobs, and the number of tardy jobs, are discussed in Sects. 4 and 5. Section 6 summarizes our results and provides suggestions for future research.

2 Problem statements and preliminary properties

The RPDm problem is defined as follows. Consider a set of jobs that is beforehand separated into m disjoint subsets

$\mathcal{J}_k = \{J_{k,1}, \ldots, J_{k,n_k}\}$ of n_k jobs for $k \in \{1, \ldots, m\}$. The subset $\mathcal{J}_k$ has to be processed on the corresponding dedicated machine M_k, and the processing sequence on each machine is known a priori and fixed. Before the project starts, a pool of v_0 units of the single-type resource is provided for processing the jobs. Each job $J_{k,h} \in \mathcal{J}_k$ is characterized by processing length $p_{k,h}$, weight $w_{k,h}$, due date $d_{k,h}$, resource requirement $\alpha_{k,h}$, and resource yield $\beta_{k,h}$. Job $J_{k,h}$ requires and immediately consumes $\alpha_{k,h}$ units of the resource from the pool to commence its processing, and on its completion time, it produces and immediately returns $\beta_{k,h}$ units of the resource to the resource pool. The contribution of job $J_{k,h}$ is defined by $\delta_{k,h} = \beta_{k,h} - \alpha_{k,h}$, which can take any sign. The problem is to determine a schedule so as to optimize one of several considered regular performance measures subject to the constraint that the schedule is feasible with respect to the resource availability.

To deal with the RPD problem subject to predetermined job sequences, we first discuss some preliminary properties. Jobs on different machines can be simultaneously executed only if their corresponding dedicated machines are free and the resource level is sufficient to support the processing of these jobs. Denote the resource level at time $t \geq 0$ under a particular schedule σ by $v_t(\sigma)$, the starting time of job $J_{k,h}$ by $s_{k,h}(\sigma)$, and the completion time of job $J_{k,h}$ by $C_{k,h}(\sigma) = s_{k,h}(\sigma) + p_{k,h}$. The schedule σ is said to be *feasible* if and only if at any time $t \in [0, \sum_k \sum_{J_{k,h} \in \mathcal{J}_k} p_{k,h}]$

$$v_t(\sigma) = v_0 + \sum_k \sum_{\{J_{k,h} \in \mathcal{J}_k | C_{k,h}(\sigma) \leq t\}} \delta_{k,h} - \sum_k \sum_{\{J_{k,h} \in \mathcal{J}_k | s_{k,h}(\sigma) \leq t < C_{k,h}(\sigma)\}} \alpha_{k,h} \geq 0. \quad (1)$$

In Eq. (1), the second term of the right-hand side is the total contribution already made by the jobs completed not later than time t, and the last term is the total amount of the resource already consumed by the jobs that are currently under processing at time t. Therefore, the composition of $v_t(\sigma)$ is interpreted as that the resource level at time t is equal to the initial resource level plus the total amount of the resource contributed by the completed jobs and minus the total amount of the resource consumed by the jobs still under processing at time t. If the resource level at some time t is negative, then the schedule is infeasible. To make the problem rigorously defined, we have to give a comparatively precise definition of $v_t(\sigma)$. Consider a specific time point t at which one or two jobs are completed and one or two jobs start processing. We consider the change of resource level around the point t with a microscopic view. The completed jobs deposit the produced resource into the pool, and thus, the resource level raises. Next, the coming jobs acquire and immediately consume the resource from the pool, and the resource level drops. Therefore, $v_t(\sigma)$ *specifies the resource level at the exact moment right after the coming jobs consume the resource they need*. The order of events is reflected in the ordering of three terms in Eq. (1).

The goal of the studied problem is to find a schedule which is feasible with respect to the initial resource level such that a particular performance measure is optimized. An objective function is called *regular* if it is non-decreasing in job completion times. For any regular objective function, we have the following property of optimal schedules in the RPD problem.

Lemma 1 *For the RPD problem with any regular objective function, there exists an optimal schedule, if any, in which any job starting time either equals 0 or coincides with the completion time of another job on some machine.*

Proof Without loss of generality, we assume that in an optimal schedule σ there exists a job $J_{1,i}$ with $s_{1,i}(\sigma) > 0$ and no job completes at time $s_{1,i}(\sigma)$. Let $t = \max\{0, \max_{k,h}\{C_{k,h}(\sigma) | C_{k,h}(\sigma) < s_{1,i}(\sigma)\}\}$. Since no job completes during the interval $[t, s_{1,i}(\sigma)]$, the feasibility of σ ensures $v_t(\sigma) \geq \alpha_{1,i}$. Therefore, job $J_{1,i}$ can be shifted backward to start at time t without altering the feasibility and optimality of σ. Repeating the above shifting process, if necessary, we obtain the optimal schedule as specified in the lemma. □

Lemma 1 also ensures that we can ignore the schedules that have all machines simultaneously idle at some time point in the scheduling span. That is, only semi-active schedules are considered.

Now we proceed to the studied RPD2 problem with fixed sequences. Assume without loss of generality that the fixed processing sequence on machine M_k is $(J_{k,1}, J_{k,2}, \ldots, J_{k,n_k})$ for $k = 1, 2$ and at least one feasible schedule exists for the studied problem. Consider a sub-schedule π for two sub-sequences of jobs $(J_{1,i}, J_{1,i+1}, \ldots, J_{1,n_1})$ and $(J_{2,j}, J_{2,j+1}, \ldots, J_{2,n_2})$, $i \in \{1, \ldots, n_1\}$, $j \in \{1, \ldots, n_2\}$. The objective function value of sub-schedule π can be determined by the completion times in $\bigcup_{i \leq h \leq n_1}\{C_{1,h}(\pi)\} \cup \bigcup_{j \leq h \leq n_2}\{C_{2,h}(\pi)\}$. If we prefix a non-negative lead time ℓ to the sub-schedule π, as shown in Fig. 1, its objective value can be calculated with $\bigcup_{i \leq h \leq n_1}\{C_{1,h}(\pi) + \ell\} \cup \bigcup_{j \leq h \leq n_2}\{C_{2,h}(\pi) + \ell\}$ and is denoted by $\gamma(\pi, \ell)$ for some addressed performance metric γ.

Property 1 *For any two sub-schedules π and π', $\gamma(\pi, 0) \leq \gamma(\pi', 0)$ holds if and only if $\gamma(\pi, \ell) \leq \gamma(\pi', \ell)$ holds for any constant $\ell \geq 0$, where $\gamma \in \{C_{\max}, \sum w_j C_j, L_{\max}\}$.*

Property 1 can be easily verified by virtue of the equality $\gamma(\pi, \ell) = \gamma(\pi, 0) + c\ell$, where $c = 1$ for $\gamma \in \{C_{\max}, L_{\max}\}$, or $c = \sum_{h=i}^{n_1} w_{1,h} + \sum_{h=j}^{n_2} w_{2,h}$ for $\gamma = \sum w_j C_j$.

In the next sections, we present a dynamic programming framework whose applicability to the studied problems stems

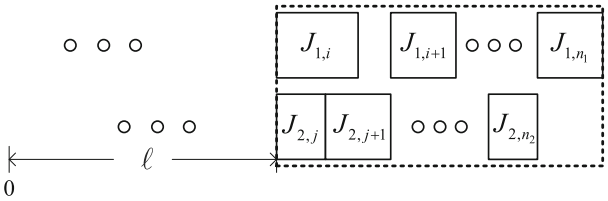


Fig. 1 Lead time ℓ followed by a sub-schedule π

from Lemma 1. Then, Property 1 leads to the polynomial-time solvability of the three performance measures, including the makespan, total weighted completion time, and maximum lateness, with a backward recursive approach.

3 Makespan, total weighted completion time, and maximum lateness

Considering a performance metric $\gamma \in \{C_{\max}, \sum w_j C_j, L_{\max}\}$, we define function $g(i, j)$, $1 \le i \le n_1$, $1 \le j \le n_2$, as the optimal objective value for scheduling the two sub-sequences of jobs $(J_{1,i}, J_{1,i+1}, \ldots, J_{1,n_1})$ and $(J_{2,j}, J_{2,j+1}, \ldots, J_{2,n_2})$, subject to the resource level $v_0 + \sum_{h=1}^{i-1} \delta_{1,h} + \sum_{h=1}^{j-1} \delta_{2,h}$. The sub-schedule attaining the objective value $g(i, j)$ is denoted by $\pi^*(i, j)$.

Property 1 implies that the sub-schedule $\pi^*(i, j)$ *dominates* all other sub-schedules in the same state (i, j) in the sense that it has the minimum value among those of all sub-schedules in this state. In addition, the backward recursion ensures that the processing span of $\pi^*(i, j)$ does not affect the scheduling decision of the unscheduled jobs $\{J_{1,1}, \ldots, J_{1,i-1}\} \cup \{J_{2,1}, \ldots, J_{2,j-1}\}$. These two observations justify the principle of optimality in the designed dynamic program, which recursively calculates the values $g(i, j)$ for $1 \le i \le n_1$, $1 \le j \le n_2$. The optimal schedule is $\pi^*(1, 1)$ and can be obtained through a simple backtracking procedure. To facilitate further presentation, define $p_{k,[i:j]} = \sum_{h=i}^{j} p_{k,h}$, $w_{k,[i:j]} = \sum_{h=i}^{j} w_{k,h}$ and $\delta_{k,[i:j]} = \sum_{h=i}^{j} \delta_{k,h}$ for $k = 1, 2$. Dummy jobs J_{k,n_k+1} and $J_{k,0}$ with $p_{k,n_k+1} = w_{k,n_k+1} = \alpha_{k,n_k+1} = \beta_{k,n_k+1} = p_{k,0} = w_{k,0} = \alpha_{k,0} = \beta_{k,0} = 0$ and $d_{k,n_k+1} = d_{k,0} = \infty$ for $k = 1, 2$ are used to define the boundary conditions.

Lemma 1 suggests that an semi-active optimal schedule can be split into several disjoint schedule blocks by slicing it at any point where either both machines start new jobs or one machine starts a new job while the other machine remains idle. Thus, sub-schedule $\pi^*(i, j)$ can be constructed as a concatenation of appropriate schedule blocks, as demonstrated in Fig. 2. In backward recursion, we augment a sub-schedule by prefixing a schedule block to it. The corresponding sub-schedule $\pi^*(i, j)$ can be obtained by considering the augmentation with three types of schedule blocks (cf. Fig. 3) prefixed:

1. Prefix $\pi^*(i+1, j)$ with the schedule block consisting of a single job $J_{1,i}$;
2. Prefix $\pi^*(i, j+1)$ with the schedule block consisting of a single job $J_{2,j}$;
3. Prefix $\pi^*(i'+1, j'+1)$ with the schedule block $B(i, j, i', j')$ comprising two overlapping sub-sequences of jobs $(J_{1,i}, \ldots, J_{1,i'}), i \le i' \le n_1$, and $(J_{2,j}, \ldots, J_{2,j'})$, $j \le j' \le n_2$. The schedule block $B(i, j, i', j')$ is called an ij-head block, which satisfies the following conditions:

   c.1 Job $J_{1,i}$ and job $J_{2,j}$ start at the same time to lead the schedule block;
   c.2 No idle time exists between any two adjacent jobs;
   c.3 Except for $J_{1,i}$ and $J_{2,j}$, no pair of machine-1 and machine-2 jobs starts at the same time;
   c.4 Job $J_{1,i'}$ starts before the completion of $J_{2,j'}$, and job $J_{2,j'}$ starts before the completion of $J_{1,i'}$.

The first two conditions **c.1** and **c.2** follow from Lemma 1, and the last two conditions **c.3** and **c.4** are imposed to ensure that a block $B(i, j, i', j')$ is minimal for inclusion. We consider only minimal blocks instead of all possible blocks to improve the running time of the designed algorithm since those non-minimal blocks can be duplicated by concatenating several appropriate minimal blocks.

Before presenting the dynamic program, we develop a pre-processing procedure for constructing the set $\mathcal{H}(i, j)$ of feasible ij-head blocks, $1 \le i \le n_1$, $1 \le j \le n_2$. Denote inequality

$$v_0 + \delta_{1,[0:i'-1]} + \delta_{2,[0:j'-1]} - \alpha_{1,i'} - \alpha_{2,j'} \ge 0$$

by condition A, which is employed to check the resource feasibility, i.e., Eq. (1). Given each pair of i and j, the procedure recursively constructs ij-head blocks by dispatching

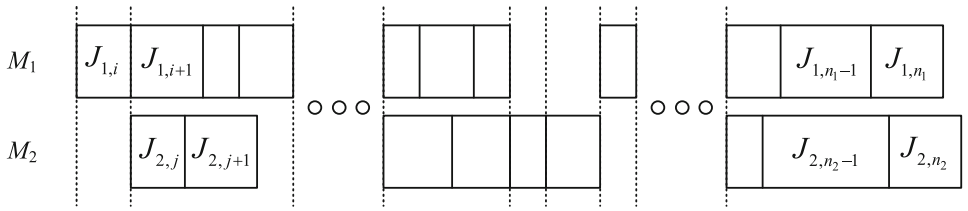


Fig. 2 Configuration of a sub-schedule $\pi^*(i, j)$

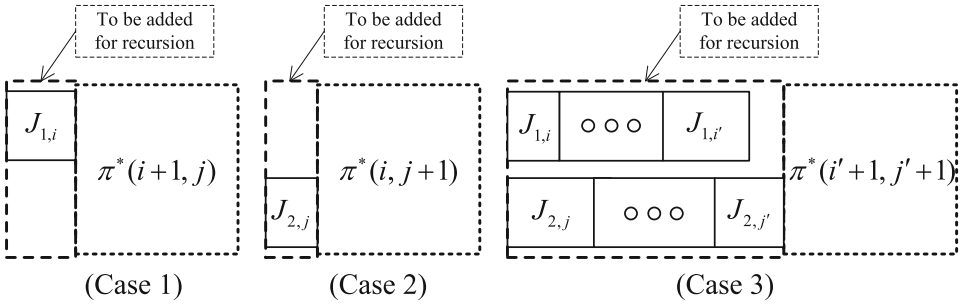


Fig. 3 Backward recursion of $\pi^*(i, j)$ with three types of schedule blocks

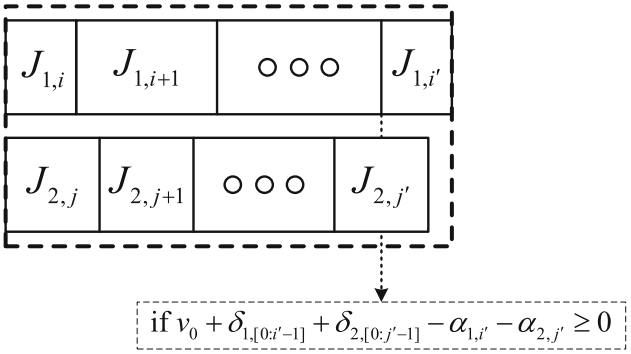


Fig. 4 Feasibility checking at the last job starting time

a job, subject to condition A, on the machine with a shorter processing span.

Construction of $\mathcal{H}(i, j)$

$\mathcal{H}(i, j) := \emptyset$; $i' := i$; $j' := j$.
while ($i' \leq n_1$ and $j' \leq n_2$ and $p_{1,[i:i']} \neq p_{2,[j:j']}$ and condition A) do
  $\mathcal{H}(i, j) := \mathcal{H}(i, j) \cup \{B(i, j, i', j')\}$; /*Add $B(i, j, i', j')$ to $\mathcal{H}(i, j)$.*/
  if $p_{1,[i:i']} < p_{2,[j:j']}$
    then $i' := i' + 1$.
  else $j' := j' + 1$.

The above procedure constructs all ij-head blocks for a given pair of i and j. In each iteration, condition A is used to ensure the feasibility of the job that starts last within block $B(i, j, i', j')$ (cf. Fig. 4). The procedure directly checks condition **c.3** with inequality $p_{1,[i:i']} \neq p_{2,[j:j']}$. Moreover, the order of appending successive jobs ensures that condition **c.4** holds. The procedure terminates when one of these conditions is violated. Note that starting with $\mathcal{H}(i, j) := \emptyset$, we augment $\mathcal{H}(i, j)$ by appending one job in each iteration. The number of iterations cannot exceed the number of jobs, i.e., $|\mathcal{H}(i, j)| \leq n_1 + n_2$. The running time of the procedure is linear in the number of unscheduled jobs for each pair of i and j.

Based upon the proposed structure of schedule blocks and the pre-processing procedure, the dynamic programming algorithm denoted by BDP- 1 is given as follows.

Algorithm BDP-1

Initialization and recursion:
for each i from $n_1 + 1$ to 1 and each j from $n_2 + 1$ to 1 do

$$g(i, j) := \begin{cases} z_0, & \text{if } i = n_1+1 \text{ and } j = n_2+1; \\ z_1, & \text{if } i \leq n_1 \text{ and } j = n_2+1; \\ z_2, & \text{if } i = n_1+1 \text{ and } j \leq n_2; \\ \min\{z_1, z_2, z_3\}, & \text{otherwise.} \end{cases} \quad (2)$$

Goal: Calculate $g(1, 1)$.

If algorithm BDP- 1 is deployed for the objective function of makespan, we define in Eq. (2) $z_0 = 0$,

$$z_1 = \begin{cases} g(i+1, j) + p_{1,i}, & \text{if } v_0 + \delta_{1,[0:i-1]} + \delta_{2,[0:j-1]} - \alpha_{1,i} \geq 0; \\ \infty, & \text{otherwise,} \end{cases}$$

$$z_2 = \begin{cases} g(i, j+1) + p_{2,j}, & \text{if } v_0 + \delta_{1,[0:i-1]} + \delta_{2,[0:j-1]} - \alpha_{2,j} \geq 0; \\ \infty, & \text{otherwise,} \end{cases}$$

and

$$z_3 = \begin{cases} \min\limits_{B(i,j,i',j') \in \mathcal{H}(i,j)} \{g(i'+1, j'+1) + \max\{p_{1,[i:i']}, p_{2,[j:j']}\}\}, & \text{if } \mathcal{H}(i, j) \neq \emptyset; \\ \infty, & \text{otherwise.} \end{cases}$$

For the performance measure of total weighted completion time, we have $z_0 = 0$,

$$z_1 = \begin{cases} g(i+1, j) + (w_{1,[i:n_1]} + w_{2,[j:n_2+1]})p_{1,i}, & \text{if } v_0 + \delta_{1,[0:i-1]} + \delta_{2,[0:j-1]} - \alpha_{1,i} \geq 0; \\ \infty, & \text{otherwise,} \end{cases}$$

$$z_2 = \begin{cases} g(i, j+1) + (w_{1,[i:n_1+1]} + w_{2,[j:n_2]})p_{2,j}, & \text{if } v_0 + \delta_{1,[0:i-1]} + \delta_{2,[0:j-1]} - \alpha_{2,j} \geq 0; \\ \infty, & \text{otherwise,} \end{cases}$$

and

$$z_3 = \begin{cases} \min\limits_{B(i,j,i',j')\in\mathcal{H}(i,j)} \Big\{ g(i'+1, j'+1) \\ \quad + \sum\limits_{h=i}^{i'} w_{1,h} p_{1,[i:h]} + \sum\limits_{h=j}^{j'} w_{2,h} p_{2,[j:h]} \\ \quad + (w_{1,[i'+1:n_1+1]} + w_{2,[j'+1:n_2+1]}) \\ \quad \times \max\{p_{1,[i:i']}, p_{2,[j:j']}\} \Big\}, & \text{if } \mathcal{H}(i,j) \neq \emptyset; \\ \infty, & \text{otherwise.} \end{cases}$$

For the minimization of maximum lateness, we define $z_0 = -\infty$,

$$z_1 = \begin{cases} \max\{-d_{1,i}, g(i+1, j)\} & \text{if } v_0 + \delta_{1,[0:i-1]} \\ \quad + p_{1,i}, & \quad + \delta_{2,[0:j-1]} - \alpha_{1,i} \geq 0; \\ \infty, & \text{otherwise,} \end{cases}$$

$$z_2 = \begin{cases} \max\{-d_{2,j}, g(i, j+1)\} & \text{if } v_0 + \delta_{1,[0:i-1]} \\ \quad + p_{2,j}, & \quad + \delta_{2,[0:j-1]} - \alpha_{2,j} \geq 0; \\ \infty, & \text{otherwise,} \end{cases}$$

and

$$z_3 = \begin{cases} \min\limits_{B(i,j,i',j')\in\mathcal{H}(i,j)} L(i,j,i',j'), & \text{if } \mathcal{H}(i,j) \neq \emptyset; \\ \infty, & \text{otherwise,} \end{cases}$$

where

$$L(i,j,i',j') = \max\left\{ \max_{i\leq h\leq i'} (p_{1,[i:h]} - d_{1,h}), \max_{j\leq h\leq j'} (p_{2,[j:h]} - d_{2,h}), \right. \\ \left. g(i'+1, j'+1) + \max\{p_{1,[i:i']}, p_{2,[j:j']}\} \right\}.$$

Algorithm BDP- 1 consists of four parts. Part 1, i.e., the first line in Eq. (2), performs the initialization by setting $g(n_1+1, n_2+1) = z_0$. Part 2 (the second line in Eq. (2)) implements the recursion for the objective function values of boundary states (i, n_2+1), $1 \leq i \leq n_1$, and no job of $\mathcal{J}_2$ is involved in the constructed sub-schedule $\pi^*(i, n_2+1)$. Similarly, the recursion down to $g(n_1+1, j)$, $1 \leq j \leq n_2$, is executed in part 3 (the third line in Eq. (2)), and no job of $\mathcal{J}_1$ is involved in $\pi^*(n_1+1, j)$. Part 4 (the fourth line in Eq. (2)) performs the recursion for general $g(i, j)$, $1 \leq i \leq n_1$, $1 \leq j \leq n_2$. Formulae z_1, z_2, and z_3 correspond to Cases 1, 2, and 3, respectively, in Fig. 3.

As for the running time, constructing all sets $\mathcal{H}(i, j)$ for $1 \leq i \leq n_1$ and $1 \leq j \leq n_2$ requires $O(n_1 n_2(n_1+n_2))$ time. In algorithm BDP- 1 which iterates $O(n_1 n_2)$ states, the calculation for each state belonging to Part 1, 2, or 3 takes a constant time, and that belonging to Part 4 requires $O(n_1+n_2)$ time due to the recursion in z_3. Therefore, the overall running time is $O(n_1 n_2(n_1+n_2))$.

Theorem 1 *The* RPD2 *problem subject to fixed sequences for the performance metric* $\gamma \in \{C_{\max}, \sum w_j C_j, L_{\max}\}$ *is solvable in* $O(n_1 n_2(n_1+n_2))$ *time.* □

We note that for the simplified scenario where all jobs have an identical processing time the type-3 schedule block is $B(i, j, i, j)$ and no recursion is required in z_3, thus reducing the running time to $O(n_1 n_2)$ time. This result can be generalized to the m-machine case, where each possible scenario for the constructed sub-schedule in a considered state is exactly associated with a non-empty subset of all the leading jobs of the m sub-sequences. Instead of only three possible scenarios (cf. Fig. 3) for the two-machine case, the number of possible scenarios for the generalized m-machine case is $2^m - 1$, and the running time is $O(2^m \prod_{k=1}^{m} n_k)$.

4 Total weighted tardiness and weighted number of tardy jobs

While the three objective functions considered in the previous section are solvable in polynomial time, the due-date-related objective functions $\sum w_j T_j$ and $\sum w_j U_j$ are however hard to solve. In this section, we will show, by a reduction from the PARTITION problem, that the fixed-sequence RPD2 problem for minimizing the total weighted tardiness or the weighted number of tardy jobs is NP-hard.

PARTITION:
INSTANCE: A positive integer B and a set of items $A = \{1, 2, \ldots, u\}$ with positive integer weights x_j, $j \in A$, such that $\sum_{j=1}^{u} x_j = 2B$.
QUESTION: Is there a partition of set A into subsets A_1 and A_2 such that $\sum_{j\in A_1} x_j = \sum_{j\in A_2} x_j = B$?

Theorem 2 *The* RPD2 *problem subject to fixed sequences for the performance metric* $\gamma = \sum w_j T_j$ *is NP-hard in the ordinary sense.*

Proof We construct an instance of $4u$ regular jobs and an enforcer job. For each $j \in A$, create four jobs, two on machine M_1 and two on machine M_2, as follows:

Jobs	$\alpha_{k,h}$	$\beta_{k,h}$	$\delta_{k,h}$	$p_{k,h}$	$d_{k,h}$	$w_{k,h}$
$J_{1,2j-1}$	j	j	0	$3uBx_j$	∞	1
$J_{1,2j}$	0	1/2	1/2	$3uBx_j + x_j$	∞	1
$J_{2,2j-1}$	j	j	0	$3uBx_j + x_j$	$(9uB+1)\sum_{r=0}^{j-1} x_r + (3uB+1)x_j$	1
$J_{2,2j}$	0	1/2	1/2	$3uBx_j$	∞	1

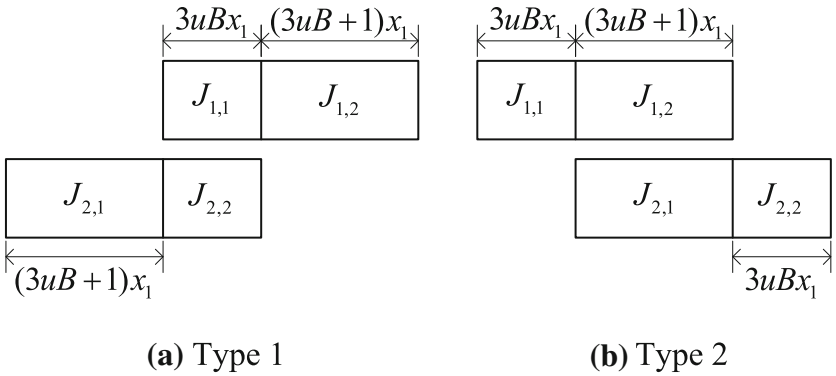


Fig. 5 The two possible arrangements of block 1

(a) Type 1 **(b)** Type 2

A dummy parameter $x_0 = 0$ is defined to facilitate the presentation. The enforcer job $J_{2,2u+1}$ is defined by $\alpha_{2,2u+1} = \beta_{2,2u+1} = u+1$, $p_{2,2u+1} = 1$, $d_{2,2u+1} = 18uB^2 + 3B + 1$, $w_{2,2u+1} = 3uB^2 + uB + 1$. The initial resource level is $v_0 = 1$. The processing sequences on both machines are

$$(J_{1,1}, J_{1,2}|J_{1,3}, J_{1,4}|\ldots, |J_{1,2u-1}, J_{1,2u}),$$
$$(J_{2,1}, J_{2,2}|J_{2,3}, J_{2,4}|\ldots, |J_{2,2u-1}, J_{2,2u}|J_{2,2u+1}),$$

where vertical bars delimit jobs created from different elements of A. We will prove that the answer to the PARTITION problem is affirmative if and only if there is a feasible schedule of the fixed-sequence RPD2 problem for $\gamma = \sum w_j T_j$ with total weighted tardiness not greater than $3uB^2 + uB$.

A feasible sub-schedule of the four jobs $J_{1,2j-1}$, $J_{1,2j}$, $J_{2,2j-1}$, $J_{2,2j}$ defined by element j of A is called *block* j. Owing to the resource constraint, any two consecutive blocks on the Gantt chart are disjoint, i.e., they do not overlap. Consider the block 1, i.e., the four jobs $J_{1,1}$, $J_{1,2}$, $J_{2,1}$, $J_{2,2}$, and its arrangement. As jobs $J_{1,1}$ and $J_{2,1}$ compete for the only one unit of the resource, they cannot be processed simultaneously. Block 1 may be one of the following two types of semi-active schedules.

Type 1: Job $J_{2,1}$ is processed first, and jobs $J_{1,1}$, $J_{2,2}$ are processed simultaneously. Then, job $J_{1,2}$ is scheduled last. This case is demonstrated in Fig. 5a.
In this case, job $J_{2,1}$ completes at $3uBx_1 + x_1$, resulting in no tardiness. The block length of this type is $9uBx_1 + 2x_1$.
Type 2: Job $J_{1,1}$ is processed first, and jobs $J_{1,2}$, $J_{2,1}$ are processed simultaneously. Then, job $J_{2,2}$ is scheduled last. This case is demonstrated in Fig. 5b.

In this case, job $J_{2,1}$ completes at $3uBx_1 + (3uBx_1 + x_1)$, resulting in the tardiness of $3uBx_1$. The block length of this type is $9uBx_1 + x_1$, which is shorter than that of Type 1 by x_1.

Let subsets A_1 and A_2 constitute a partition of set A. Construct a schedule by concatenating blocks $1, 2, \ldots, u$. In addition, the enforcer job $J_{2,2u+1}$ is scheduled last. The blocks corresponding to the elements of set A_1 are scheduled in Type 1, and those corresponding to the elements of set A_2 are scheduled in Type 2. It can be easily verified that all regular jobs can be successfully processed, and that the total processing length of the u blocks is $(9uB^2+2B)+(9uB^2+B)$. At the completion of the last regular job, the resource level is $u+1$. Therefore, the enforcer job can be processed, and the constructed schedule is feasible. We calculate the total weighted tardiness of the schedule as follows. If block j is of Type 1 rather than Type 2, then it delays the processing of all blocks $j' > j$ by x_j, incurring potential extra total weighted tardiness by at most $(u-j)x_j < ux_j$. If all blocks are of Type 2, the total weighted tardiness of all regular jobs corresponding to A_2 is minimized and calculated as $\sum_{j\in A_2} 3uBx_j = 3uB^2$. Therefore, the total weighted tardiness of the constructed schedule is bounded from above by $3uB^2 + \sum_{j\in A_1} ux_j = 3uB^2 + uB$.

For the "*if*" part of the proof, we assume that there is a feasible schedule with total weighted tardiness not greater than $3uB^2 + uB$. Since the weight of the enforcer job is larger than the threshold, it must be started not later than $18uB^2+3B$. Let A_1 (respectively, A_2) be the set of indices of Type 1 blocks (respectively, Type 2 blocks). Let $\sum_{j\in A_1} x_j = B + \epsilon$ for $\epsilon \in [-B, B]$. The processing length of all blocks is given by

$$\sum_{j\in A_1}(9uBx_j + 2x_j) + \sum_{j\in A_2}(9uBx_j + x_j) = 18uB^2 + 3B + \epsilon. \quad (3)$$

The total weighted tardiness of all regular jobs is *at least*

$$\sum_{j\in A_2} 3uBx_j = 3uB(B-\epsilon). \quad (4)$$

By Eq. (3), if $\epsilon > 0$, then the enforcer job will be tardy, inducing a tardiness penalty exceeding the threshold. On the other hand, if $\epsilon < 0$, then by Eq. (4) the total weighted tardiness of all regular jobs is at least $3uB(B+1) > 3uB^2 + uB$, a contradiction to the assumption of the total weighted

tardiness. Therefore, ϵ must be zero, implying that subsets A_1 and A_2 constitute a partition of set A. □

Theorem 3 *The* RPD2 *problem subject to fixed sequences for the performance metric* $\gamma = \sum w_j U_j$ *is NP-hard in the ordinary sense.*

Proof The proof is similar to that of Theorem 2 with slight modifications. Job parameters are given as follows with modified parameters shown in boldface:

Jobs	$\alpha_{k,h}$	$\beta_{k,h}$	$\delta_{k,h}$	$p_{k,h}$	$d_{k,h}$	$w_{k,h}$
$J_{1,2j-1}$	j	j	0	$3uBx_j$	∞	1
$J_{1,2j}$	0	1/2	1/2	$3uBx_j + x_j$	∞	1
$J_{2,2j-1}$	j	j	0	$3uBx_j + x_j$	$(9uB+1)\sum_{r=0}^{j-1} x_r + (3uB+1)x_j + \boldsymbol{B}$	$\boldsymbol{x_j}$
$J_{2,2j}$	0	1/2	1/2	$3uBx_j$	∞	1

We set $x_0 = 0$ and define the enforcer job $J_{2,2u+1}$ by $\alpha_{2,2u+1} = \beta_{2,2u+1} = u+1$, $p_{2,2u+1} = 1$, $d_{2,2u+1} = 18uB^2 + 3B + 1$, $w_{2,2u+1} = \boldsymbol{B+1}$. The initial resource level is also $v_0 = 1$. It can be shown that the answer to the PARTITION problem is affirmative if and only if there is a feasible schedule of the fixed-sequence RPD2 problem for $\gamma = \sum w_j U_j$ with the weighted number of tardy jobs not greater than B. □

The above two theorems confirm the hardness in producing an optimal schedule for the performance measure $\gamma \in \{\sum w_j T_j, \sum w_j U_j\}$. We proceed to the development of a pseudo-polynomial-time dynamic programs. If job $J_{k,h}$ completes at time t in a particular schedule, its tardiness is given by $T_{k,h}(t) = \max\{0, t - d_{k,h}\}$ and its tardiness status is indicated by the binary variable

$$U_{k,h}(t) = \begin{cases} 1, & \text{if } t > d_{k,h}; \\ 0, & \text{otherwise.} \end{cases}$$

The notion of schedule blocks proposed in Sect. 3 is followed. Note that Property 1 does not hold for the two objective functions considered in this section. An extra state variable ℓ specifying the lead time before a considered sub-schedule is required in the backward dynamic program because the optimal decision in each recursion cannot be made without a fixed lead time.

To facilitate the discussion, we denote $\ell_1(i,j) = \max\{p_{1,[0:i-1]}, p_{2,[0:j-1]}\}$ and $\ell_2(i,j) = p_{1,[0:i-1]} + p_{2,[0:j-1]}$. Define function $g(i,j,\ell)$ for $1 \le i \le n_1$, $1 \le j \le n_2$ and $\ell_1(i,j) \le \ell \le \ell_2(i,j)$, as the optimal objective value for scheduling the two sub-sequences $(J_{1,i}, J_{1,i+1}, \ldots, J_{1,n_1})$ and $(J_{2,j}, J_{2,j+1}, \ldots, J_{2,n_2})$, subject to the condition that the lead time in front of the constructed sub-schedule is exactly ℓ. The sub-schedule attaining the value of $g(i,j,\ell)$ is denoted by $\pi^*(i,j,\ell)$. The same three types of schedule blocks in Fig. 3 are used to obtain a sub-schedule $\pi^*(i,j,\ell)$. The backward dynamic program, denoted by BDP- 2, is a variant adapted from BDP-1. It recursively calculates for the values of $g(i,j,\ell)$. The optimal objective value is given by $g(1,1,0)$ and an optimal schedule can be constructed by backtracking the recursion.

Algorithm BDP-2

Initialization and recursion:
`for` each i from n_1+1 to 1, each j from n_2+1 to 1 and each ℓ from $\ell_1(i,j)$ to $\ell_2(i,j)$ `do`

$$g(i,j,\ell) := \begin{cases} 0, & \text{if } i = n_1+1 \text{ and } j = n_2+1; \\ z_1, & \text{if } i \le n_1 \text{ and } j = n_2+1; \\ z_2, & \text{if } i = n_1+1 \text{ and } j \le n_2; \\ \min\{z_1, z_2, z_3\}, & \text{otherwise.} \end{cases} \tag{5}$$

Goal: Calculate $g(1,1,0)$.

Define function $f_{k,h}(t) = w_{k,h}T_{k,h}(t)$ (or $f_{k,h}(t) = w_{k,h}U_{k,h}(t)$) for the performance metric $\gamma = \sum w_j T_j$ (or $\gamma = \sum w_j U_j$). In Eq. (5), we denote

$$z_1 = \begin{cases} g(i+1, j, \ell + p_{1,i}) + f_{1,i}(\ell + p_{1,i}), & \text{if } v_0 + \delta_{1,[0:i-1]} + \delta_{2,[0:j-1]} - \alpha_{1,i} \ge 0; \\ \infty, & \text{otherwise,} \end{cases}$$

$$z_2 = \begin{cases} g(i, j+1, \ell + p_{2,j}) + f_{2,j}(\ell + p_{2,j}), & \text{if } v_0 + \delta_{1,[0:i-1]} + \delta_{2,[0:j-1]} - \alpha_{2,j} \ge 0; \\ \infty, & \text{otherwise,} \end{cases}$$

and

$$z_3 = \begin{cases} \min\limits_{B(i,j,i',j') \in \mathcal{H}(i,j)} L(i,j,i',j'), & \text{if } \mathcal{H}(i,j) \ne \emptyset; \\ \infty, & \text{otherwise,} \end{cases}$$

where

$$L(i,j,i',j') = g(i'+1, j'+1, \ell + \max\{p_{1,[i:i']}, p_{2,[j:j']}\}) + \sum_{h=i}^{i'} f_{1,h}(\ell + p_{1,[i:h]}) + \sum_{h=j}^{j'} f_{2,h}(\ell + p_{2,[j:h]}).$$

Algorithm BDP- 2 iterates $O(n_1 n_2 \min\{p_{1,[1:n_1]}, p_{2,[1:n_2]}\})$ states, each of which requires $O(n_1+n_2)$ time for enumerating the blocks $B(i,j,i',j')$. The total running time is $O(n_1 n_2 (n_1+n_2) \min\{p_{1,[1:n_1]}, p_{2,[1:n_2]}\})$, which depends on the schedule span and is thus pseudo-polynomial.

Theorem 4 *The RPD2 problem subject to fixed sequences for the performance metric $\gamma \in \{\sum w_j T_j, \sum w_j U_j\}$ can be solved in $O(n_1 n_2 (n_1 + n_2) \min\{p_{1,[1:n_1]}, p_{2,[1:n_2]}\})$ time.* □

For the simplified scenario where all jobs have a common processing time, the type-3 schedule block $B(i, j, i, j)$ implies that no recursion in z_3 is required. Moreover, the lead time is a multiple of the common processing time, so the range for the recursive loop with respect to ℓ in BDP- 2 can be replaced with the interval $\big[\max\{i-1, j-1\}, i+j-2\big]$. Therefore, the overall running time is $O(n_1 n_2 \min\{n_1, n_2\})$. The algorithm can be generalized for the case of m dedicated machines with a running time of $O(2^m \prod_{k=1}^{m} n_k \sum_{k=1}^{m} n_k)$.

5 Number of tardy jobs

This section is dedicated to the minimization of the number of tardy jobs ($\sum U_j$). A dynamic program adopting the idea of schedule blocks given in Sect. 3 is designed in forward recursion. Define function $g(i, j, \lambda)$ for $1 \le i \le n_1$, $1 \le j \le n_2$, and $0 \le \lambda \le i + j$ as the minimal processing span for scheduling the two sub-sequences $(J_{1,1}, J_{1,2}, \ldots, J_{1,i})$ and $(J_{2,1}, J_{2,2}, \ldots, J_{2,j})$, subject to the condition that the number of tardy jobs is at most λ. The sub-schedule attaining $g(i, j, \lambda)$ is denoted by $\pi^*(i, j, \lambda)$. After the forward recursion for the values of $g(i, j, \lambda)$, the optimal objective value is given by $\min\{\lambda \mid g(n_1, n_2, \lambda) < \infty\}$. The notion of introducing the considered performance measure $\sum U_j$ as a state variable stems from its characteristic that the number of possible objective values is exactly the number of jobs. A forward dynamic programming algorithm, denoted by FDP, is outlined as follows, and the forward recursion is depicted in Fig. 6.

Algorithm FDP

Initialization and recursion:
`for` each i from 0 to n_1 and each j from 0 to n_2 `do`

$g(i, j, -1) := \infty$;
$g(i, j, i + j + 1) := g(i, j, i + j)$.

`for` each i from 0 to n_1, each j from 0 to n_2 and each λ from 0 to $i + j$ `do`

$$g(i, j, \lambda) := \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0; \\ z_1, & \text{if } i \ge 1 \text{ and } j = 0; \\ z_2, & \text{if } i = 0 \text{ and } j \ge 1; \\ \min\{z_1, z_2, z_3\}, & \text{otherwise.} \end{cases} \quad (6)$$

Goal: Calculate $\min_{0 \le \lambda \le n_1 + n_2} \{\lambda \mid g(n_1, n_2, \lambda) < \infty\}$.

In Eq. (6), we define

$$z_1 = \begin{cases} L_1(i, j, \lambda), & \text{if } v_0 + \delta_{1,[0:i-1]} + \delta_{2,[0:j]} - \alpha_{1,i} \ge 0; \\ \infty, & \text{otherwise,} \end{cases}$$

where

$$L_1(i, j, \lambda) = \min \begin{cases} g(i-1, j, \lambda - 1) + p_{1,i}, & \text{if } g(i-1, j, \lambda - 1) + p_{1,i} > d_{1,i}; \\ g(i-1, j, \lambda) + p_{1,i}, & \text{if } g(i-1, j, \lambda) + p_{1,i} \le d_{1,i}, \end{cases}$$

$$z_2 = \begin{cases} L_2(i, j, \lambda), & \text{if } v_0 + \delta_{1,[0:i]} + \delta_{2,[0:j-1]} - \alpha_{2,j} \ge 0; \\ \infty, & \text{otherwise,} \end{cases}$$

where

$$L_2(i, j, \lambda) = \min \begin{cases} g(i, j-1, \lambda - 1) + p_{2,j}, & \text{if } g(i, j-1, \lambda - 1) + p_{2,j} > d_{2,j}; \\ g(i, j-1, \lambda) + p_{2,j}, & \text{if } g(i, j-1, \lambda) + p_{2,j} \le d_{2,j}, \end{cases}$$

and

$$z_3 = \min_{1 \le i' \le i, 1 \le j' \le j} \big\{ L_3(i', j', i, j, \lambda) \mid B(i', j', i, j) \in \mathcal{H}(i', j') \big\},$$

where

$$L_3(i', j', i, j, \lambda) = \min_{0 \le \lambda' \le \min\{i'+j'-2, \lambda\}} \begin{cases} g(i'-1, j'-1, \lambda') & \text{if condition B;} \\ \quad + \max\{p_{1,[i':i]}, p_{2,[j':j]}\}, & \\ \infty, & \text{otherwise.} \end{cases}$$

And condition B is set as

$$\sum_{h=i'}^{i} U_{1,h} \big(g(i'-1, j'-1, \lambda') + p_{1,[i':h]} \big) + \sum_{h=j'}^{j} U_{2,h} \big(g(i'-1, j'-1, \lambda') + p_{2,[j':h]} \big) \le \lambda - \lambda'.$$

The recursion in algorithm FDP iterates $O(n_1 n_2 (n_1 + n_2))$ states, each of which requires at most $O(n_1 n_2 (n_1 + n_2))$ time. The goal step needs to perform $O(n_1 + n_2)$ comparisons, each of which requires a constant time. Thus, the overall running time is $O(n_1^2 n_2^2 (n_1 + n_2)^2)$.

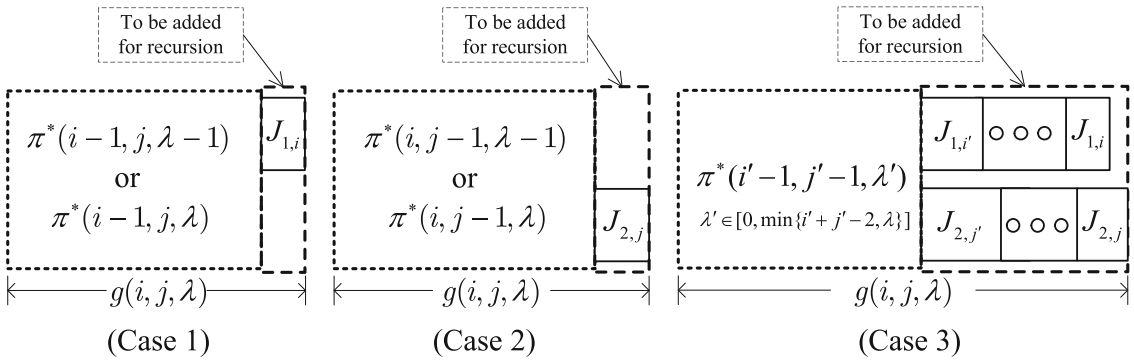


Fig. 6 Forward recursion of $\pi^*(i, j, \lambda)$ with three types of schedule blocks

Theorem 5 *The fixed-sequence* RPD2 *problem for minimizing the number of tardy jobs, i.e., $\gamma = \sum U_j$, is solvable in $O(n_1^2 n_2^2 (n_1+n_2)^2)$ time.* □

6 Concluding remarks

This paper addressed the relocation problem on two parallel dedicated machines, where the jobs are processed in predetermined sequences. The complexity hierarchy of the RPD2 problem subject to fixed processing sequences is provided in Fig. 7. For the case with a common processing time, all the considered regular performance measures are polynomially solvable. Given that the common-processing-time case is considered for m parallel dedicated machines, algorithm BDP- 1 can also be generalized and has an $O(2^m \prod_{k=1}^{m} n_k)$ running time, where n_k is the number of jobs on machine M_k. Similarly, algorithm BDP- 2 can be generalized with an $O(2^m \prod_{k=1}^{m} n_k \sum_{k=1}^{m} n_k)$ running time. Both running times are polynomial when the number of parallel dedicated machines m is constant. Although the problem settings of the study could be deemed limited concerning real-world applications, the motivation from the algorithmic perspective that the developed techniques and complexity results are potentially exploitable in the solving of the general relocation problem shall be highlighted.

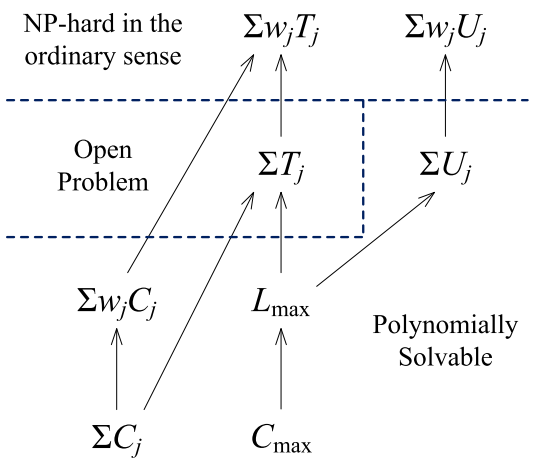


Fig. 7 The complexity hierarchy of the fixed-sequence RPD2 problem

Future study could consider the development of a polynomial-time algorithm or NP-hardness proof for the open problem with $\gamma = \sum T_j$. In an attempt to develop a polynomial-time algorithm, more structural properties of the objective function $\sum T_j$ need to be explored. As for the NP-hardness proof, the assumption of fixed sequences reduces the flexibility in manipulating the jobs. As a sequel, conventional concepts deployed to prove NP-hardness may not work. The intractability of the case with $\gamma \in \{\sum w_j T_j, \sum w_j U_j\}$ also suggests the development of approximation algorithms. It would be of special interest to know the approximability, e.g., if approximation schemes exist for these two ordinary NP-hard resource-constrained problems.

A final point concluding this paper concerns the relationship between the relocation problem and the inventory-constrained scheduling, which arises from numerous applications in manufacturing and logistics management. The inventory-constrained scheduling problems, depending on specific problem definitions, could be special cases or generalizations of the relocation problem. Truck scheduling at a transshipment terminal, for example, could be formulated as the problem with resource constraints (Briskorn et al. 2010), where the resource (i.e., inventory) is depleted (by loading trucks) and replenished (by uploading trucks) both at the job completion times, i.e., $\alpha_{k,h} = 0$ and $\beta_{k,h}$ is unrestricted in sign, and the inventory level is required to be non-negative throughout. In the project scheduling subject to inventory constraints (Laborie 2003; Neumann and Schwindt 2002), the precedence network of jobs, variable job processing times, and minimum and maximum inventory levels could be considered. An investment project, for example, is funded with initial capital (i.e., resource), and each job is initiated and completed with a positive or negative cash flow. Further research could be conducted in extending the solution

approaches developed in the relocation problem to the relevant scheduling problems with inventory constraints.

Acknowledgments The authors are grateful to the anonymous reviewers for their constructive comments that have improved earlier versions of this paper. Lin was supported in part by the Ministry of Science and Technology of Taiwan under Grants NSC-97-2923-H-009-001-MY3 and NSC-102-2923-H-009-001-MY3. Kononov was partially supported by the Russian Foundation of Humanities under Grant RFH-NSC-13-02-10002.

References

Amir, A., & Kaplan, E. H. (1988). Relocation problems are hard. *International Journal of Computer Mathematics*, *25*(2), 101–110.

Briskorn, D., Choi, B.-C., Lee, K., Leung, J., & Pinedo, M. (2010). Complexity of single machine scheduling subject to nonnegative inventory constraints. *European Journal of Operational Research*, *207*(2), 605–619.

Cheng, T. C. E., Lin, B. M. T., & Toker, A. (2000). Makespan minimization in the two-machine flowshop batch scheduling problem. *Naval Research Logistics*, *47*(2), 128–144.

Hwang, F. J., Kovalyov, M. Y., & Lin, B. M. T. (2012). Total completion time minimization in two-machine flow shop scheduling problems with a fixed job sequence. *Discrete Optimization*, *9*(1), 29–39.

Hwang, F. J., Kovalyov, M. Y., & Lin, B. M. T. (2014). Scheduling for fabrication and assembly in a two-machine flowshop with a fixed job sequence. *Annals of Operations Research*, *217*(1), 263–279.

Hwang, F. J., & Lin, B. M. T. (2011). Coupled-task scheduling on a single machine subject to a fixed job sequence. *Computer & Industrial Engineering*, *60*(4), 690–698.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, *1*(1), 61–68.

Kaplan, E. H. (1986). Relocation models for public housing redevelopment programs. *Environment and Planning B: Planning and Design*, *13*(1), 5–19.

Kaplan, E. H., & Amir, A. (1988). A fast feasibility test for relocation problems. *European Journal of Operational Research*, *35*(2), 201–206.

Kaplan, E. H., & Berman, O. (1988). OR hits the heights: Relocation planning at the orient heights housing project. *Interfaces*, *18*(6), 14–22.

Kononov, A. V., & Lin, B. M. T. (2006). On relocation problems with multiple identical working crews. *Discrete Optimization*, *3*(4), 366–381.

Kononov, A. V., & Lin, B. M. T. (2010). Minimizing the total weighted completion time in the relocation problem. *Journal of Scheduling*, *13*(2), 123–129.

Laborie, P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, *143*(2), 151–188.

Lin, B. M. T., & Cheng, T. C. E. (2006). Two-machine flowshop scheduling with conditional deteriorating second operations. *International Transactions in Operational Research*, *13*(2), 91–98.

Lin, B. M. T., & Hwang, F. J. (2011). Total completion time minimization in a 2-stage differentiation flowshop with fixed sequences per job type. *Information Processing Letters*, *111*(5), 208–212.

Neumann, K., & Schwindt, C. (2002). Project scheduling with inventory constraints. *Mathematical Methods of Operations Research*, *56*(3), 513–533.

Ng, C. T., & Kovalyov, M. Y. (2007). Batching and scheduling in a multi-machine flow shop. *Journal of Scheduling*, *10*(6), 353–364.

Shafransky, Y. M., & Strusevich, V. A. (1998). The open shop scheduling problem with a given sequence of jobs on one machine. *Naval Research Logistics*, *45*(7), 705–731.