Journal of Industrial and Management Optimization

https://www.aimspress.com/journal/jimo

JIMO, 22(10): 5227–5246.
DOI:10.3934/jimo.2026180
Received: 13 March 2026
Revised: 07 September 2026
Accepted: 11 September 2026
Published: 21 September 2026

Research article

Makespan minimization on two parallel dedicated machines under resource constraints

Huai-Che Hong[1], **Elaine Yi-Ling Wu**[2] **and Bertrand M.T. Lin**[1,3,*]

[1] Institute of Information Management, Institute of Hospital and Health Care Administration, National Yang Ming Chiao Tung University, Hsinchu 300093, Taiwan

[2] Department of Information Management, National Dong Hwa University, Hualien 974301, Taiwan

[3] International College, Department of Computer Science and Information Management, Providence University, Taichung 433303, Taiwan

* **Correspondence:** Email: bmtlin@nycu.edu.tw.

Abstract: This paper investigates a strongly NP-hard two-machine scheduling problem where the objective is to minimize the makespan. The system consists of two machines, each assigned a dedicated set of jobs. Processing is subject to a common resource pool of a single-type resource. A key challenge lies in the dynamic resource requirement: each job must acquire a specific amount of the resource to commence its processing, and it releases a potentially unequal amount back to the pool upon completion. To model this complex resource flow and processing interdependence, we first develop a mixed-integer linear programming model. Recognizing the problem's strong NP-hardness, we embed an existing polynomial-time dynamic programming algorithm, which was designed for two given processing sequences a priori, within two newly designed search-based meta-heuristics. We conclude by presenting extensive computational tests that rigorously evaluate the performance characteristics of these algorithms.

Keywords: resource constraints; relocation project; parallel dedicated machines; NP-hardness; meta-heuristics

Mathematics Subject Classification: 90B35, 90C11, 90C59

1. Introduction

This paper studies a resource-constrained scheduling problem that originates from a relocation project. Resource constraints are usually categorized as renewable or nonrenewable. The former reflects to the situations where at the completion of an activity, the resources that the activity has

acquired are released for later activities. Renewable resources such as personnel, pallets, hoisters, and heavy-duty equipments/vehicles are among this type. Nonrenewable resources usually refer to the consumables, such as energy, inventory of goods, and storage space, subject to a certain amount to upper limits within different planning periods. For resource-constrained project planning and scheduling, there are excellent reviews, surveys, and books, which include, but are not limited to, Artigues et al. [1], Brucker et al. [2], Hartmann & Briskorn [3], Herroelen et al. [4], Issa & Tu [5] and Özdamar & Ulusoy [6].

In this paper, we consider a new variant of renewable resource constraints introduced and formulated by Kaplan [7] from a municipal redevelopment project in east Boston. At the beginning of the redevelopment project, a set of buildings was to be redeveloped. The reconstruction process consists of two operations, demolition and erection. Before a building was torn down, all of its tenants had to be evacuated and temporarily housed. Therefore, sufficient housing units were required for settling the tenants whose residential areas were being redeveloped. After redevelopment, the housing units of the new buildings could be used to settle newly evacuated tenants. The new capacity of a building is not necessarily the same as its original capacity. For example, the site of some buildings could be planned to be a park or a recreation center. Furthermore, tenants could choose to reside in any available building and were not required to return to the reconstructed building at their original location. The objective was to determine a feasible redevelopment schedule that successfully processes all buildings without leaving any tenants unsettled during the whole schedule. The problem was first addressed by Kaplan and Amir [8].

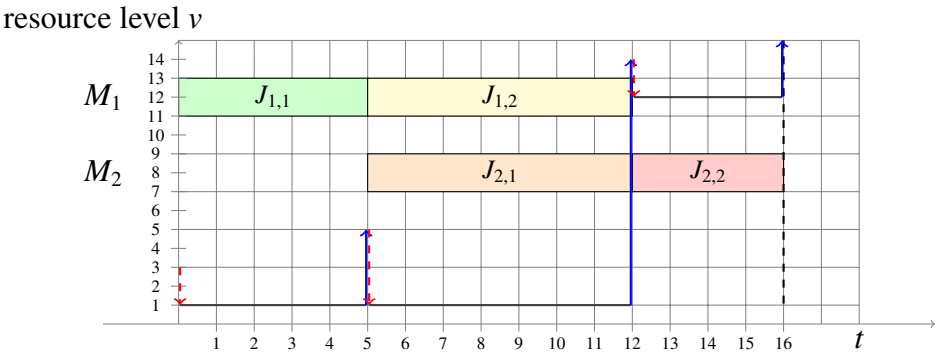
The studied relocation scheduling problem in the context of parallel dedicated machines is defined as follows. We will introduce the basic relocation problem in the next section. Two machines M_1 and M_2 are dedicated to processing two disjoint sets of jobs $\mathcal{J}_1 = \{J_{1,1}, \ldots, J_{1,n_1}\}$ and $\mathcal{J}_2 = \{J_{2,1}, \ldots, J_{2,n_2}\}$, where n_1 and n_2 are the numbers of jobs respectively assigned the two machines. Job $J_{i,j} \in \mathcal{J}_i$ for $i \in \{1, 2\}$ is characterized by processing time $p_{i,j} > 0$, resource requirement $a_{i,j} \geq 0$, and resource yield $b_{i,j} \geq 0$. An initial level of v_0 units of a single-type resource are provided in a common repository for supporting the processing of all the given jobs. To start the processing of job $J_{i,j}$, the incumbent resource level should be larger than or equal to $a_{i,j}$. When job $J_{i,j}$ starts its processing, $a_{i,j}$ (≥ 0) units of the resource in the resource repository are immediately consumed. Once a job starts, it must be processed until its completion without any interruption or preemption. When job $J_{i,j}$ is finished, it immediately produces and deposits $b_{i,j}$ (≥ 0) units back to the resource repository. The *contributions* job $J_{i,j}$ makes to the resource pool when it is successfully processed, and is defined by $\delta_{i,j} = b_{i,j} - a_{i,j}$, which can take any sign. At any time, a machine can process at most one job. When the resource is sufficient, the processing of two jobs could overlap on the two machines. The problem is to determine a feasible schedule such that all jobs can be successfully finished in the shortest time, i.e., the makespan is minimum. Here, feasibility refers to the condition that no job is left unprocessed due to a shortage of the resource. Following the commonly adopted three-field notation of Graham et al. [9], we denote the studied problem by $PD2|a_{i,j}, b_{i,j}|C_{\max}$, where the first field $PD2$ indicates the processing floor of two parallel dedicated machines, the second field $a_{i,j}, b_{i,j}$ stands for the resource parameters, and $C_{\max}$ indicates the objective of makespan minimization.

To illustrate the problem definition, we consider the following example. Given are two sets of jobs $\mathcal{J}_1, \mathcal{J}_2$ to be processed on machine M_1 and machine M_2, respectively. Each set contains two jobs. Parameter values of the jobs are given in Figure 1a. We make two feasible schedules for a comparison.

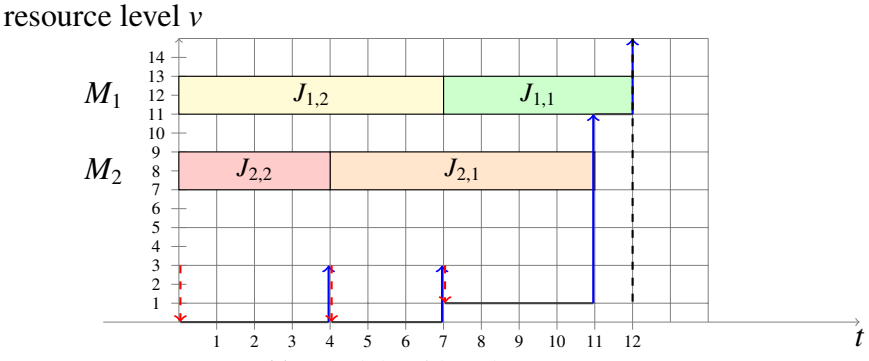
In the Gantt chart of Figure 1b, machine M_1 processes job $J_{1,1}$ first. The resource level soon drops to 1 because job $J_{1,1}$ consumes $a_{1,1} = 2$ units of the resource. Note that machine M_2 cannot process any of its jobs because of resource insufficiency. At the completion of $J_{1,1}$, it produces and deposits $b_{1,1} = 4$ units into the resource pool and the resource level rises to 5. The incumbent resource level is sufficient to support the simultaneous commencement of jobs $J_{1,2}$ and $J_{2,1}$. Then, job $J_{2,2}$ follows. The resulting makespan $C_{\max}$ is 16. If the jobs are processed with better overlapping arrangement, we can obtain a schedule with a makespan of 12 as shown in Figure 1c. The optimization problem is to construct a feasible schedule such that all jobs are finished at the earliest possible time. Note that the total contribution of all jobs $\sum_{i=1}^{2}\sum_{j=1}^{2}\delta_{i,j} = 2+2+7+1 = 12$ is fixed once the input is given. Any feasible schedule ends up with a resource level $3+\sum_{i=1}^{2}\sum_{j=1}^{2}\delta_{i,j} = 15$.

Job	$J_{1,1}$	$J_{1,2}$	$J_{2,1}$	$J_{2,2}$
$a_{i,j}$	2	1	3	2
$b_{i,j}$	4	3	10	3
$p_{i,j}$	5	7	7	4
Initial resource level $v_0 = 3$				

(a) Numerical instance of two job sets.



(b) Schedule with makespan 16.



(c) Schedule with makespan 12.

Figure 1. Gantt charts of two example schedules.

This study makes three contributions. First, we strengthen the known complexity status of $PD2|a_{i,j}, b_{i,j}|C_{\max}$: the problem remains strongly NP-hard even if the processing sequence on one machine is given and all jobs have positive contributions. This result sharpens the boundary between

tractable and intractable cases and justifies the two solution approaches that follow. Second, we propose the first mixed integer linear programming model for the problem, which yields exact solutions and benchmarks for small instances. Third, because the problem is intractable, we design two meta-heuristics that embed the dynamic programming algorithm of Lin et al. [10], originally developed for fixed processing sequences, as a fitness-evaluation subroutine. This embedding turns an exact algorithm for a restricted setting into an efficient building block for the general problem, and is the main methodological novelty of this study.

We organize the paper as follows. Section 2 reviews the base relocation problem and proliferated researches that are relevant to the studied problem. Preliminary properties will also be addressed. In Section 3, we formulate an integer programming model with time-indexed decision variables. Section 4 presents two meta-heuristics that incorporate a polynomial-time dynamic programming algorithm as a subroutine to produce sub-optimal solutions. A computational study for evaluating the performances of the proposed approaches is conducted and the results are discussed. We conclude our work in Section 6 and suggest future research subjects.

2. Related researches and preliminaries

The relocation problem was proposed and studied by Kaplan [7]. The application context is a relocation project for municipal transformations in east Boston. The base model is to determine the minimum initial budget for guaranteeing the existence of a feasible plan. Each building has a capacity of α_j residential units before the redevelopment. To redevelop building j, we need α_j temporary housing units to accommodate the residents. At its completion, the site has a new capacity of β_j units. Residents may relocate to reside in a different site. The base problem is to determine a feasible construction sequence of all buildings subject to the limited budget of temporary housing. An optimization view is to determine the minimum budget level for ensuring feasibility of the project. Kaplan [8] established the equivalence between the minimization of initial resource level in the relocation project and the minimization of makespan (or total idle time on the second-stage machine) in two-machine flow shop scheduling. As a consequence of the equivalence, determining the minimum resource provision in the base relocation problem can be solved by deploying the well-known Johnson's algorithm [11].

To reflect the realistic scenarios raised by the Boston Housing Authority in the Orient Heights redevelopment project, Kaplan and Berman [12] extended the mathematical formulation. Kaplan and Berman [12] proposed the parallel-machine scheduling model in which multiple buildings are allowed to be redeveloped if the available resource is sufficient for housing all tenants of the overlapped development. For the parallel-machine environment, Amir and Kaplan [13] proved that the relocation problem with two identical working crews is NP-hard. Kononov and Lin [14] gave a strong NP-hardness proof of a restricted case, in which two parallel identical dedicated machines are assigned to process unit-execution-time (UET) jobs and the contribution of all jobs are positive. They gave a non-approximability proof for the case where the number of machines is unbounded and all jobs have UET and their contributions are negative. Furthermore, approximation algorithms and their associated performance-ratio analysis were proposed for the cases with unbounded as well as bounded numbers of machines. Since the two-machine relocation problem with UET jobs is computationally challenging, Lin et al. [10] considered the case where the processing sequences on

the two machines are given and fixed. They proposed polynomial-time dynamic programming algorithms for three regular objective functions, i.e. makespan, total weighted completion time, and total tardiness. The $PD2|a_{i,j}, b_{i,j}|C_{\max}$ problem considered in this study does not involve the assumption about fixed job sequences on the machines.

Several extensions from the base model have been studied in the literature. Under a single machine setting, Lin and Tseng [15] considered the problem with deadline constraints and gave a complexity analysis result. There are several research findings about the relocation problem with due dates considered. Considering a common due date, Lin and Tseng [16] proposed an approach to maximize the resource level within a given duration. In the work of Lin and Liu [17], they extended this problem by introducing the concept of generated due date introduced by Hall [18]. They proved that the relocation problem of maximizing the reward with generalized due date is strongly NP-hard and proposed a branch-and-bound algorithm equipped with two dominance rules and two lower bounds. Sevastyanov et al. [19] introduced release dates into the relocation problem to minimize the makespan. They analyzed the complexity status of several different problem settings and proposed a multi-parametric dynamic programming algorithm to produce optimal schedules. It is worth noting that while the minimum initial resource level for single-machine relocation scheduling can be determined in polynomial time via Johnson's algorithm, the problem of minimizing makespan is strongly NP-hard. Su and Lin [20] proposed lower bounds and dynamic dominance rules for the design of branch-and-bound algorithms, implemented with various branching strategies. The performances of the proposed properties and strategies were appraised through computational tests. Cheng et al. [21] generalized the single-machine model into a two-machine flow shop, where the first machine is for demolishing buildings and the second machine for erecting new buildings. In the sense of resource dynamics, jobs processed on machine 1 consume the resources and on machine 2 recycle the resource. They investigated optimality properties of the new model, discussed the time complexities of several cases, and derived a complexity hierarchy of these cases. Lo and Lin [22] developed a branch-and-bound algorithm and several heuristics to solve the resource recycling setting.

The application of the studied problem $PD2|a_{i,j}, b_{i,j}|C_{\max}$ in the real world can be found in housing revitalization projects. Several buildings need to be reconstructed. In order to accelerate the reconstruction process, two identical working crews are responsible for the whole projects. Since the moving cost of crews and equipment is relatively high, buildings are partitioned into two disjunctive sets in view of geological proximity. The goal is to find a feasible schedule such that the time span used for completing the whole project is minimum. Another interpretation is about the scheduling two neighboring docks for container ships. The docks share a common container yard, whose vacancy is the common resource pool. A ship can unload containers only when there is available vacancy, and it releases vacancy when it uploads containers for departure.

2.1. Preliminary properties

The study on the $PD2|a_{i,j}, b_{i,j}|C_{\max}$ problem starts with preliminary properties. First, we note that jobs on different machines can be executed simultaneously if their corresponding dedicated machines are free and the resource level is sufficient to support the processing of these multiple jobs. Let σ be

some schedule, under which we define the following notation:

$$\begin{aligned} v_t(\sigma)&: \text{the resource level at time } t \geq 0 \text{ under } \sigma;\\ S_{i,j}(\sigma)&: \text{the starting time of job } J_{i,j};\\ C_{i,j}(\sigma) = S_{i,j}(\sigma) + p_{i,j}&: \text{the completion time of job } J_{i,j}. \end{aligned}$$

To simplify the presentation, the notation of schedule σ will be omitted when no confusion would arise.

Definition: Schedule σ is *feasible* if and only if at any time point, the resource level is nonnegative, i.e., the inequality

$$v_t(\sigma) \quad = \quad v_0 + \sum_{i=1}^{2} \sum_{\substack{J_{i,j}\in\mathcal{J}_i \\ C_{i,j}(\sigma)\leq t}} \delta_{i,j} - \sum_{i=1}^{2} \sum_{\substack{J_{i,j}\in\mathcal{J}_i \\ S_{i,j}(\sigma)\leq t<C_{i,j}(\sigma)}} a_{i,j} \geq 0. \tag{2.1}$$

is satisfied for any time point $t \in [0, \sum_{i=1}^{2} \sum_{J_{i,j}\in\mathcal{J}_i} p_{i,j})$. □

The second term in Eq (2.1) indicates the total contribution made by the jobs that are already completed before or at time t. The last term is the total amount of the resource occupied by the jobs that have started but are still under processing at time t. The nonnegativity inequality requires the schedule σ has a processing plan in which the resource level is nonnegative at any time of the planning horizon.

The following result connects the relocation problem to the classical two-machine flow shop scheduling of makespan minimization.

Theorem 1. *[8] For the single-machine relocation scheduling problem, the minimum initial resource level v_0 that guarantees the existence of feasible schedules for $\mathcal{J}$ equals to the minimum total idle time on machine 2 in the classical two-machine flow shop scheduling of makespan minimization, if we let a_j and b_j be the processing times of job j on the two machines.*

On a single-machine in the relocation scheduling problem, we have a set of jobs $\mathcal{J} = \mathcal{J}^+ \cup \mathcal{J}^-$. $\mathcal{J}^+ = \{J_j \mid \delta_j = b_j - a_j \geq 0\}$ consists of jobs whose contributions are nonnegative while $\mathcal{J}^- = \{J_j \mid \delta_j = b_j - a_j < 0\}$ contains jobs whose contributions are negative. The resource requirement of the Johnson's sequence, denoted by $\sigma_{\text{JS}} = \sigma_{\mathcal{J}^+} \oplus \sigma_{\mathcal{J}^-}$, where $\sigma_{\mathcal{J}^+}$ is the sequence in nondecreasing order of a_j for all $J_j \in \mathcal{J}^+$ and $\sigma_{\mathcal{J}^-}$ is the sequence in nonincreasing order of b_j for all $J_j \in \mathcal{J}^-$. Therefore, no feasible solutions would exist if and only if the initial resource level v_0 is less than the minimum resource requirement of the job sequence arranged by Johnson's algorithm.

We return to the current subject of two-machine relocation scheduling. The following results indicate that $PD2|a_{i,j}, b_{i,j}|C_{\max}$ is intractable even if for the very restricted case.

Theorem 2. *[10] Minimizing the makespan on two dedicated machines is strongly NP-hard even if all jobs have a unit execution time ($p_{i,j} = 1$) and nonnegative contributions ($\delta_{i,j} \geq 0$).*

Theorem 3. *[10] Minimizing the makespan on two dedicated machines subject to the condition that the processing sequences on both machines are known a priori is solvable in $O(n_1 n_2 (n_1 + n_2))$ for objective functions $\gamma \in \{C_{\max}, L_{\max}, \sum w_j C_j\}$.*

The solution algorithm of Lin et al. [10] deploys a backward dynamic programming approach based upon the concept of job blocks. The scenarios of Theorem 3 have the processing sequences on both machines given and fixed. This paper studies the same problem without this assumption. That is, each machine is associated with a subset of jobs whose processing sequence is to be determined. Furthermore, the processing intervals of all jobs on both machines have to be properly arranged to ensure schedule feasibility addressed in Eq (2.1).

We first prove that the problem remains strongly NP-hard even if the assumption about two fixed processing sequences is removed for one machine.

Theorem 4. *Makespan minimization in PD2*$|a_{i,j}, b_{i,j}|C_{\max}$ *is strongly NP-hard even if (1) the processing sequence on one machine is given, and (2)* $\delta_{i,j} > 0$ *for all* $i \in \{1, 2\}, J_{i,j} \in \mathcal{J}_i$.

Proof: The proof is carried out by a reduction from 3-Partition, which is known to be strongly NP-hard (Garey and Johnson [23]).
3-Partition: Given are a set A of $3q$ elements $\{1, 2, \ldots, 3q\}$ and an integer E such that each $j \in A$ has a size e_j that satisfies $E/4 < e_j < E/2$ and that the total size is $\sum_{j=1}^{3q} e_j = qE$. Is there a partition $A_1, A_2, \ldots, A_q$ of the set A such that $\sum_{j \in A_l} e_j = E, 1 \le l \le q$?

Without loss of generality, each e_j is a positive integer, and thus $E > 1$. Given an instance of 3-Partition, we create a scheduling instance of two disjoint sets with $n_1 = 3q$ and $n_2 = q$ as follows:

$$\mathcal{J}_1 : a_{1,j} = 0, b_{1,j} = e_j, p_{1,j} = e_j, \quad 1 \le j \le 3q;$$

$$\mathcal{J}_2 : a_{2,j} = E + (j-1)E^2, b_{2,j} = jE^2, p_{2,j} = E, \quad 1 \le j \le q.$$

The initial resource level is defined as $v_0 = E$. The fixed sequence of jobs is given on specified of machine M_2 in order of their indices $(J_{2,1}, J_{2,2}, \ldots, J_{2,n_2})$. Since $E > 1$, $\delta_{2,j} = b_{2,j} - a_{2,j} = E^2 - E > 0$.

The proof is to confirm that the answer to the 3-Partition instance is affirmative if and only if there is a feasible schedule whose makespan is exactly qE.

(i) Assume that $A_1, A_2, \ldots, A_q$ is a desired partition. We schedule the jobs corresponding to the elements of A_1 in the interval $[0, E)$ on machine M_1. Machine M_2 processes job $J_{2,1}$ in interval $[0, E)$ as well. The three jobs on machine M_1 do not require any resource, and job $J_{2,1}$ on machine M_2 consumes E units of the resource. The four jobs are successfully processed. At time E, the resource level becomes $v_E = E^2 + E$ due to the contributions of the four jobs. In the interval $[E, 2E)$, the next three jobs on machine M_1 and job $J_{2,2}$ can be similarly processed, resulting in $v_{2E} = 2E^2 + E$. Continuing the arrangement, we come up with a feasible schedule in the interval $[0, qE)$ without any idle time.

(ii) Given a feasible schedule with a makespan of qE, we reason that a partition of 3-Partition exists. The makespan is qE, implying that no idle time is permitted on both machines. Let A_1 be the set of the first three jobs on machine M_1 overlapping with job $J_{2,1}$. We consider two disjoint cases:

- $\sum_{j \in A_1} e_j > E$: In this case, at time E, only two jobs of A_1 can complete before time E, resulting in the resource level $v_E < E^2 + E$ because any $e_j < E/2$. The resource level is insufficient to commence job $J_{2,2}$ immediately after job $J_{2,1}$.
- $\sum_{j \in A_1} e_j < E$: In such a case, a fourth job on machine M_1 will start before time E and complete after time E. Therefore, at time E, the resource level $v_E < E^2 + E$ is insufficient for job $J_{2,2}$.

The above discussion leads to the fact that $\sum_{j\in A_1} e_j = E$. Following the same line of reasoning, we come up with the set subsets $A_2, \ldots, A_q$ satisfying $\sum_{j\in A_2} e_j = \cdots = \sum_{j\in A_q} e_j = E$. A desired partition is subsequently implied. □

The above theorem affirms that $PD2|a_{i,j}, b_{i,j}|C_{\max}$ is computationally challenging. The remaining sections will present different approaches to tackle the studied problem.

3. Mixed integer programming model

To better describe the problem with mathematical rigor and have a first attempt to obtain solutions, we propose a mixed integer programming model of the $PD2|a_{i,j}, b_{i,j}|C_{\max}$ problem using a time-indexed approach. For any feasible schedules, Eq (2.1) must hold at any time point. This idea can be retained in the mixed integer linear programming formulation as well.

In this formulation, we use a set of auxiliary variables v_t to keep track of the resource level at time t. Besides, because of the characteristics of dedicated machines, binary decision variables $x_{i,j,t}$ reveal the information on the starting point of jobs $J_{i,j}$. That is, $x_{i,j,t} = 1$ if job $J_{i,j}$ starts at time t, and 0, otherwise. The length of the planning horizon T is bounded within the interval $[\max_{i=1,2}\{\sum_{j=1}^{n_i} p_{i,j}\}, \sum_{i=1}^{2}\sum_{j=1}^{n_i} p_{i,j})$, where the left bound indicates that processing spans of the two machines overlap and the right limit refers to the scenarios that they are mutually disjoint. The time index t takes integer values in $\{0, 1, \ldots, T-1\}$. Since all processing times $p_{i,j}$ are positive integers, restricting starting times to integer time points incurs no loss of optimality. The formulation is described in the following, where $C_{\max}$ (makespan) and v_t (resource level at time t) are continuous variables.

Mixed integer programming model

$$\text{Minimize } C_{\max} \tag{3.1}$$

subject to

$$\sum_{t\in T} x_{i,j,t} = 1, \qquad 1 \le j \le n_i, i \in \{1,2\}; \tag{3.2}$$

$$\sum_{j=1}^{n_i} \sum_{t'\in\{t-p_{i,j}+1,\ldots,t\}} x_{i,j,t'} \le 1, \qquad t \in T, i \in \{1,2\}; \tag{3.3}$$

$$\sum_{t\in T} x_{i,j,t}(t + p_{i,j}) \le C_{\max}, \qquad i \in \{1,2\}, j \in \{1,2,\ldots,n_i\}, t \in T; \tag{3.4}$$

$$v_t = v_{t-1} + \sum_{i=1}^{2}\sum_{j=1}^{n_i} x_{i,j,(t-p_{i,j})} b_{i,j} - \sum_{i=1}^{2}\sum_{j=1}^{n_i} x_{i,j,t} a_{i,j}, \qquad t \in T; \tag{3.5}$$

$$\sum_{i=1}^{2}\sum_{j=1}^{n_i} a_{i,j} x_{i,j,t} \le v_t, \qquad t \in T; \tag{3.6}$$

$$x_{i,j,t} \in \{0,1\}, \qquad i \in \{1,2\}, j \in \{1,2,\ldots,n_i\}, t \in T; \tag{3.7}$$

$$v_t \ge 0, \qquad t \in T. \tag{3.8}$$

In the above model, constraints (3.2) ensure that the any job has exactly one starting point. Constraints (3.3) guarantee that each machine can process at most one job at any time t. Constraints (3.4) indicate that the makespan is not less than the completion time of any job. Constraints (3.5) are used to keep track of the resource level in the pool at any time. Constraints (3.6) ensure that the resource level is larger than or equal to the total resource requirements of any job that starts its processing at any time point t. Constraints (3.7) describe binary values, and constraints (3.8) ensure the nonnegativity of resource levels. Before closing this section, we note that the value of T could be reduced from $\sum_{i=1}^{2}\sum_{j=1}^{n_i} p_{i,j}$ for decreasing the number of decision variables. Given any two sequences for the two machines, we apply the dynamic programming algorithm of Lin et al. [10]. The derived solution value (makespan) can serve as an upper bound of T.

4. Metaheuristic algorithms

Since the $PD2|a_{i,j}, b_{i,j}|C_{\max}$ problem is strongly NP-hard, it could be time consuming to solve large instances. We circumvent to design approximation algorithms that are expected to produce quality solutions in a reasonable time. Recall that the problem can be solved by a polynomial-time dynamic programming algorithm if the job sequences on both machines are known (Theorem 3). Given two processing sequences on two machines, an optimal solution can be found in polynomial time. Therefore, we can iterate on different processing sequences on the two machines to find approximation solutions. We therefore fabricate the dynamic programming algorithm of Lin et al. [10] into the design of two permutation-based meta-heuristic algorithms, simulated annealing and tabu search, to produce sub-optimal solutions.

4.1. Simulated annealing

Kirkpatrick et al. [24] introduces simulated annealing (SA) to deal with combinatorial optimization problems. Annealing is a thermal process for obtaining the low energy state of a solid. The process first heats a solid metal until it is melted into a liquid and lowers its temperature gradually. Particles will randomly arrange themselves until reaching thermal equilibrium after temperature changes. Transitions between different temperatures can be regarded as a Markovian process and the transition probability from one state to another can be calculated by the Metropolis algorithm. In combinatorial optimization problems, a search algorithm creates a neighborhood of the current solution, and then moves to one of its neighbors. The process iterates until the predefined conditions are met. Such an algorithm is called local search. SA combines the concept of local search and the discrete time homogeneous Markovian process to improve solution qualities repetitively. Furthermore, deteriorating movement to inferior solution qualities is allowed probabilistically in order to escape from the trap of local optimum. The SA algorithm is presented in the following.

Algorithm 1: Simulated Annealing

Data:

- X_0: Initial Solution generated by Johnson's algorithm.
- T_0: Initial temperature generated by Hot-enough condition algorithm.
- L: Neighborhood Size.
- U: Upper limit on the number of times where solutions have not been improved continuously.
- α: Cooling rate.

$X := X_0$;
$T := T_0$;
$X^* := X_0$;
$count := 0$;
while $count < U$ **do**
  for $l = 1$ *to* L **do**
    $X' := LocalSearch(X)$ $\Delta := f(X') - f(X)$
    if $\Delta < 0$ **then**
      $X := X'$
    end
    if $f(X') < f(X^*)$ **then**
      $X^* := X'$
      $count := 0$
    else
      $count := count + 1$
    end
    if $\Delta < 0$ *or* $accept\_rate < exp(\frac{-\Delta}{T})$ **then**
      $X := X'$
    end
  end
  $T := \alpha \times T$
end
Return: X^*

There are several factors that may affect the performance and convergence of SA when solving the studied problem. These factors include the structure of solutions, the solution quality of initial solution, the procedure that generates neighborhood solutions, the initial temperature, the acceptance rate of deteriorating movement, the number of iterations, and so on. The combination of parameters mentioned above should be adjusted appropriately to generate good solutions based on problem characteristics.

In the following, explanations of the decision on the parameters listed below will be presented in greater details: structure of solution, initial solution, generation of neighborhood solutions, initial temperature, and cooling strategy.

Structure of solution:

In the algorithms we propose, the representation of solutions is two processing sequences on the two machines. By doing so, we can use the dynamic programming model in Theorem 3 to calculate the fitness function value of any solution. Figure 2 shows a part of two processing sequences.

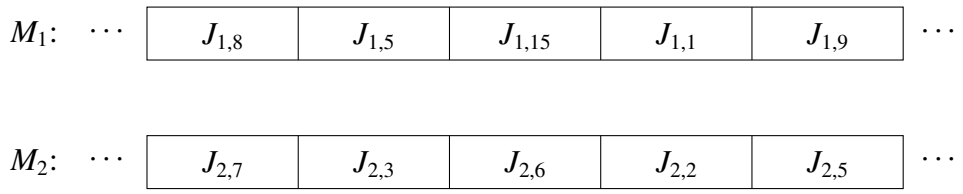


Figure 2. Example of sequence-based solution structures

Initial solution:
From Theorem 1, Johnson's sequence guarantees the minimum resource requirement of a given data in the single-machine environment. For parallel machine environment, we can also apply Johnson's algorithm to the set of jobs $\mathcal{J}_1$ and $\mathcal{J}_2$ respectively to generate a solution which respects the resource availability if the minimum resource requirement of σ_{JS}, which is generated from applying Johnson's algorithm to $\mathcal{J} = \mathcal{J}_1 \cup \mathcal{J}_2$, is less than the initial resource level. That is, if the resource requirement of σ_{JS} is equal to v_0, there will be two jobs executed on both machines for any time t and the makespan is equal to $\sum_{M_i \in \mathcal{M}} \sum_{J_{i,j} \in \mathcal{J}_i} p_{i,j}$.

Generation of neighborhood solutions:
Selection of different neighborhood definitions has direct impacts on the efficiency and effectiveness of most search algorithms. We compare three commonly adopted neighborhood definitions. The first method, called *flip*, will randomly select two consecutive jobs on one machine and exchange their positions. The second method, named *two-swap*, will randomly select two jobs on the same machine and exchange their positions. The last one is the well-known *two-opt*, which was first proposed to deal with the traveling salesperson problem. Two-opt randomly selects a sub-processing sequence on one machine and reverses this subsequence to generate a new neighbor solution. In order to find the best local search method to our problem, we conduct a preparatory computational experiment with three different neighborhood structures. In the experiment, different neighborhood structures are incorporated into simulated annealing algorithms while all parameters are all set to the same value. The initial temperatures are generated by Algorithm 2 and all neighbors of the current solution will be examined. The results of experiments are presented in Figure 3.

We introduce the notation of θ before explaining Figure 3. Let v_0^* be the minimum initial resource level for the existence of feasible schedules. If the provision of initial resource is higher, then more feasible schedules will emerge. Control parameter $\theta \geq 1$ is introduced to prescribe the initial resource level $v_0 = \theta \times v_0^*$ provided to the job instance. Because of the fact that all heuristics report the same solutions, we can focus on the comparisons between the total numbers of fitness calculated with different neighborhood structures. It can be observed that flip converged faster than the others when the instances have lower resource availability ($\theta = 1.1$), but two-opt converged faster while resource availability became larger. The number of times fitness calculated in two-opt is not worse than that of flip when higher resource availability is given. Moreover, the performance of two-swap is the worst in all situations. Therefore, we selected flip and two-opt as the neighborhood structures to be used in the meta-heuristics implementations.

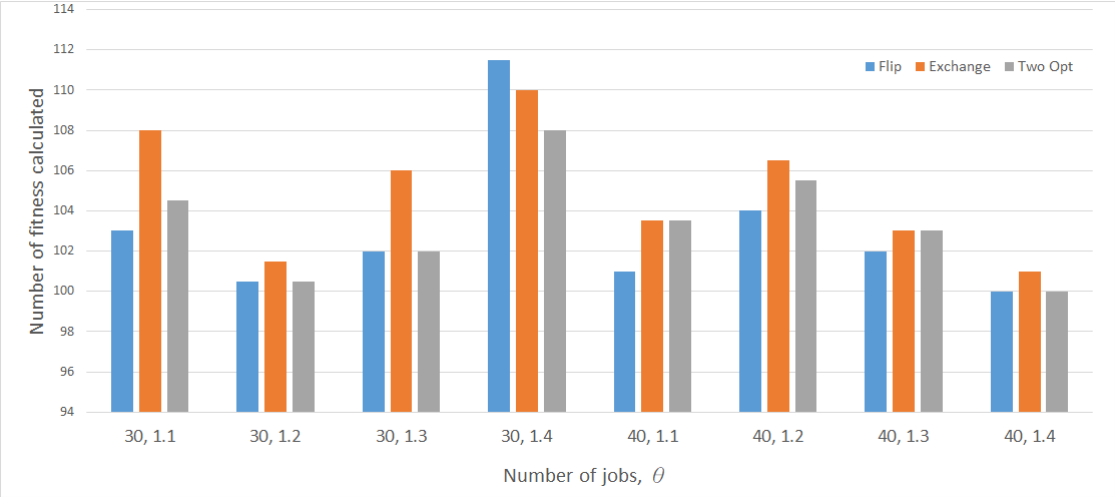


Figure 3. Comparison between different neighborhood structures.

Initial temperature:
Generally, the initial temperature should be hot enough to guarantee all movements are acceptable. However, setting the initial temperature to a higher value will cause a much longer convergence time in preparation of the problem-solving session. Therefore, we use the algorithm 2 to determine a hot-enough temperature as the initial temperature.

Algorithm 2: Initial Temperature Setting

Step 1: Sample over R random iterations and compute the values of

- n_1: the number of decreasing transitions trials.
- n_2: the number of increasing transitions trials
- Δ^+: the average increase in n_2 trails.
- r: the acceptance ratio.

Step 2:

Calculate $A = \frac{n_1 + n_2 exp(\frac{-\Delta^+}{T})}{R}$

Step 3:

if $A < 1$ **then**

$T := \frac{\Delta+}{\log(\frac{n_2}{n_2 r - n_1(1-r)})}$

end

Repeat **Steps 1 to 3** until A is close to r.

Cooling strategy:
The cooling strategy determines the way the system temperature decreases along with the solution-finding iterations. There are two rates widely adopted in the literature. The first is based on the formula $T_{k+1} = \frac{T_0}{\log(k+2)}$, which guarantees final convergence. The other one uses a geometric rate $\alpha \in (0, 1)$ to calculate $T_k = \alpha T_{k-1}, k = 1, 2, \ldots$ Since the former is time-consuming to reach convergence, we adopted the later one as the cooling strategy in our SA design.

4.2. Tabu search

Tabu search (TS) is a well-known meta-heuristic algorithm proposed by Glover [25]. The word "Tabu (or Taboo)" comes from Tongan, a language in Polynesia, and means prohibition. A TS algorithm starts from a given initial solution, then it will move iteratively from one solution to another based on the current solution until reaching the stopping criterion and then output the best solution observed during exploring. At each iteration, there is a list of candidate moves chosen from a definitive neighborhood. In order to avoid a reverse search on any examined area, a concept of adaptive memory is introduced. Those candidates that have been recently visited are marked as tabu and are not accessible to avoid cycling. A tabu list is maintained to keep track of certain tenure of tabu candidates. The detailed of the TS framework is presented in Algorithm 3.

Parameter settings

TS-based methods have been implemented in several areas of combinatorial optimization problems, and have proved to be very successful. However, parameter controlling such as the size of tabu list and aspiration criterion plays a key role in the actual performance of TS. The decision on parameter setting mentioned in the following list will be explained as in the following:

- Structure and generation of neighborhood solutions.
- Size of tabu list.
- Aspiration criteria.

Algorithm 3: Tabu search algorithm

Data:

- X_0: Initial Solution generated by Johnson's Algorithm.
- TL: Size of tabu list.
- L: Neighborhood Size.
- U: Upper limit of the number of times where solutions have not been improved continuously.
- AC: Acceptance ratio.

```
X := X_0, count := 0;
while count < U do
    for l = 1 to L do
        X' := LocalSearch(X) if X' is not tabu then
            if f(X') < f(X*) then
                X* := X';
                count := 0;
            else
                count := count + 1;
            end
            X := X';
            Put the direction of X' into tabu list;
        else
            if f(X') < f(X*) and r < AC then
                X* := X', count := 0;
            end
        end
    end
end
Result: Return: X*
```

Structure and generation of neighborhood solutions:
The encoding of solutions is the same as the encoding used in SA. As we mentioned in the previous section, dramatical changes in solution structures will lead to deterioration in solution quality. Therefore, the neighborhood generation method will be the same as in algorithm SA. That is, flip operation and two-opt operation will be adopted.

The size of tabu list:
Tabu list is one of the major components in tabu search algorithms. For each move during iterations, the characteristics of the moves are recorded in the tabu list. In the literature, tabu list length of 7 is widely adopted. Although such a setting is successful in some applications, it seems to be an inflexible approach. Glover [26] proposed the reverse elimination method (REM) to manage the tabu list used dynamically. REM follows the main purpose of tabu list so as to avoid moving to an explored solution. Therefore, at each iteration, a movement list which is used to record all moves conducted in the past is traced back to determine moves that should be set as tabu because they would lead to a solution already explored. We will call the tracing procedure a *trace*. Different from the original tabu list which sets moves tabu, we would set the complements of these moves tabu. For instance, given a solution $(x(1), x(2), ..., x(n))$ in a zero-one integer programming model, a move e is to set variable x_i from 0 to 1 (1 to 0), in the following iterations, the move $\bar{e}$ is forbidden because reversal is not allowed. In order to find out if all moves should be set tabu, a residual cancellation sequence (RCS) is built. Starting with an empty RCS, only the moves whose complement is not in the RCS are added. Otherwise, their complement in the sequence is canceled. By doing so, preventing the current solution turns back to an explored solution.

Nevertheless, as the size of movement list increases, it is time consuming for large iteration numbers to trace back the whole list. Glover [27] also suggested that a trace could be terminated after adding a move within a specified tracing step where no complement exists in the earlier entries of movement list. Such a move is named a *solo attribute*. Dammeyer and Voß [28] provided a counterexample and corrected the earlier termination scenario solo attribute by introducing a second solo attribute. That is, once we find two different solo attributes, the trace can be terminated. For the studied problem, the two types of tabu list management mentioned above are all implemented, and computational results will be presented in the following section.

Aspiration criteria:
Aspiration criteria refer to a set of criteria that allow for exception from the tabu list if one of them is fulfilled. Several ways are proposed for various purposes. We adopt two commonly used criteria in the TS algorithm for the studied problem.

1. Aspiration by default: A move with the least tabu status is allowed if all neighbors are set tabu.
2. Aspiration by objective: If the fitness function value of a move marked as tabu is better than the best known solution found thus far, then the tabu status can be released with an acceptance probability associated with the degree improvement.

5. Computational experiments

In this section, computational experiments are conducted to measure the performances of the two metaheuristics and the mixed integer programming model proposed in this study. The data generation scheme is described first and then the results of the experiments will be analyzed. Optimal solutions

are found using the off-the-shelf optimizer Gurobi. All experiments are implemented in C++ and performed on a Linux server with an i7-6800k CPU, 64GB RAM.

For all algorithms, the upper limit of numbers of times when the solution is not improved continuously U is set to 50. The number of neighbors generated in each iteration L is set to be 20% of neighbors of the current solution which are generated randomly.

5.1. Data generation schemes

All of the data generated are integer. Processing times $p_{i,k}$ of job $J_{i,k}$ were generated by rounding values from the normal distribution with $(\mu, \sigma) = (10, 3)$ to the nearest positive integer. Resource parameters of $a_{i,k}$ and $b_{i,k}$ were generated by rounding values from the normal distribution with $(\mu, \sigma) = (100, 10)$ to the nearest positive integer. The initial resource level v_0 is calculated by θV, where V is the minimum resource requirement of Johnson's sequence, and θ represents the flexibility in resource availability. The value of θ will be set to be larger than 1 in order to guarantee feasibility.

5.2. Performance of mixed integer program

We will discuss the performance of the proposed mixed integer programming model and compare it with SA and TS we implemented for this problem. For each size of jobs (n), we generate 5 independent instances. For each instance, the time limit for searching the optimal solution is 600 seconds. We set T to the makespan obtained by applying the dynamic programming algorithm of Lin et al. [10] to the two initial sequences produced by Johnson's algorithm. This bound is tighter than $\sum_{i=1}^{2} \sum_{j=1}^{n_i} p_{i,j}$. Tabu search equipped with different tabu list management strategies are also compared. We can see that:

1. Table 1 and Table 2 show the result of the mixed integer program model. The model can hardly reach the optimal solution in 10 minutes if the number of jobs is larger than 20; even though with size equal to 20, it takes time to reach the optimal solution. This result is conceivable because the number of constraints and decision variables grow largely when the size of instance grows.
2. The performances of all meta-heuristics with flip are better than the same meta-heuristics with two-opt when $\theta = 1.2$, and the number of fitness calculated with flip is larger than the number of fitness with two-opt. We can infer that the two-opt technique causes the premature situation. However, with more flexibility in resource availability, such a phenomenon does not exist.
3. TS_REM converges slower than TS_Fixed. It seems that dynamic tabu list management provides a more effective way to provide the solution with better quality.
4. SA converges slower than others and provides better solution quality than others. The hot-enough initial temperature condition provides better exploration directions.

Table 1. Comparison between mixed integer program and meta-heuristics with $\theta = 1.2$.

n	IP			SA_Flip				SA_Two-opt				TS_Fixed_Flip			
	IP	Time	Gap	ObjVal	#-f	Time	Gap	ObjVal	#-f	Time	Gap	ObjVal	#-f	Time	Gap
5	**52.0**	2.82	0.00%	**52.0**	67.2	0.00	0.00%	53.6	50.0	0.00	3.08%	52.0	50.0	0.00	0.00%
10	68.6	121.01	0.00%	68.8	97.2	0.04	0.29%	70.8	56.4	0.03	3.21%	69.4	51.0	0.03	1.17%
15	89.2	481.60	0.00%	89.2	52.0	0.12	0.00%	89.2	52.0	0.11	0.00%	89.2	52.0	0.17	0.00%
20	117.2	510.40	0.00%	117.2	54.0	0.30	0.00%	117.2	54.0	0.30	0.00%	117.2	57.6	0.43	0.00%
25	151.0	600.00	13.88%	132.6	57.6	0.65	0.00%	132.6	57.6	0.67	0.00%	132.6	70.4	0.98	0.00%

n	TS_Fixed_Two-opt				TS_REM_Flip				TS_REM_Two-opt			
	ObjVal	#-f	Time	Gap	ObjVal	#-f	Time	Gap	ObjVal	#-f	Time	Gap
5	53.6	50.0	0.00	3.08%	52.0	52.0	0.00	0.00%	53.6	50.0	0.00	3.08%
10	70.8	58.2	0.04	3.21%	69.4	51.0	0.04	1.17%	70.8	56.4	0.04	3.21%
15	89.2	52.0	0.18	0.00%	89.2	52.0	0.20	0.00%	89.2	52.0	0.20	0.00%
20	117.2	55.2	0.51	0.00%	117.2	61.2	0.53	0.00%	117.2	56.4	0.53	0.00%
25	132.6	56.0	1.09	0.00%	132.6	57.6	1.10	0.00%	132.6	56.0	1.13	0.00%

Table 2. Comparison between mixed integer program and meta-heuristics with $\theta = 1.4$.

n	IP			SA_Flip				SA_Two-opt				TS_Fixed_Flip			
	Obj	Time	Gap (%)	Obj	#-f	Time	Gap (%)	Obj	#-f	Time	Gap (%)	Obj	#-f	Time	Gap (%)
5	46.0	0.76	0.00%	46.0	51.2	0.00	0.00 %	46.0	50.0	0.00	0.00%	46.0	50.0	0.00	0.00%
10	58.4	127.42	0.00%	58.4	51.0	0.04	0.00%	58.4	51.0	0.04	0.00%	58.4	51.0	0.05	0.00%
15	91.4	381.60	0.00%	91.4	52.0	0.13	0.00%	91.4	52.0	0.12	0.00%	91.4	55.2	0.18	0.00%
20	111.6	247.03	0.00%	111.6	54.0	0.31	0.00%	111.6	54.0	0.34	0.00%	111.6	54.0	0.49	0.00%
25	138.8	600.00	4.68	132.6	57.6	0.65	0.00%	132.6	57.6	0.67	0.00%	132.6	64.0	0.98	0.00

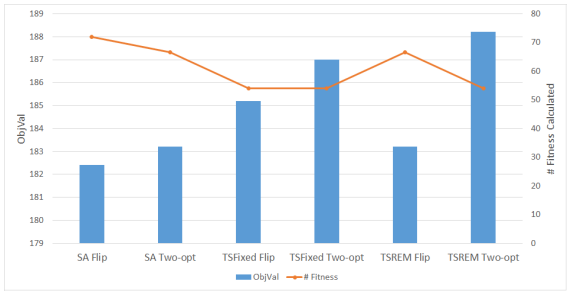
n	TS_Fixed_Two-opt				TS_REM_Flip				TS_REM_Two-opt			
	Obj	#-f	Time	Gap (%)	Obj	#-f	Time	Gap (%)	Obj	#-f	Time	Gap (%)
5	46.0	50.0	0.00	0.00%	46.0	52.0	0.00	0.00%	46.0	50.0	0.00	0.00%
10	58.4	51.0	0.05	0.00%	58.4	52.2	0.06	0.00%	58.4	51.0	0.07	0.00%
15	91.4	52.0	0.18	0.00%	91.4	52.0	0.23	0.00%	91.4	52.0	0.21	0.00%
20	111.6	54.0	0.55	0.00%	111.6	55.2	0.57	0.00%	111.6	58.8	0.57	0.00%
25	132.6	57.6	1.09	0.00%	132.6	60.8	1.10	0.00%	132.6	59.2	1.13	0.00%

5.3. Performance of meta-heuristic algorithms

Since the mixed integer programming model's run time grows exponentially with instance size, we compare SA, TS_Fixed, and TS_REM on large-scale datasets. Figure 4 to Figure 7 show the results.

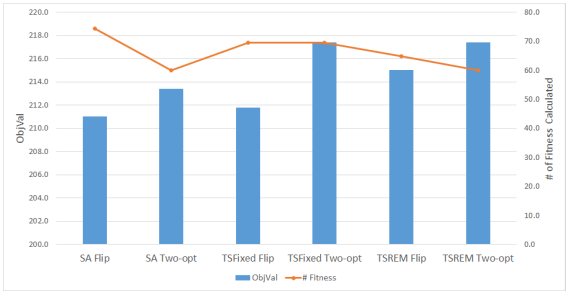
1. We can see that TS_fixed takes more iterations to convergence than TS_REM, and accordingly, its performance is better than TS_REM. This phenomenon is not consistent with small-scale datasets. It can be inferred that as the number of neighborhood size grows, the possibility to move to an explored solution is reduced, therefore, the premature situation will not happen as often.
2. As the scale of instances increases, the performances of algorithms with the two-opt technique are worse than the same algorithms with the flip technique. We can infer that the more dramatic changes in solution structure, the more possibility to reach infeasible solutions.
3. SA outperforms other algorithms in small-scale datasets, however, the results are not the same in large-scale datasets. It cannot escape from the local optimum efficiently with hot-enough initial temperature condition when the search space grows. Nevertheless, the performance of SA is still favorable.

(a) $\theta = 1.2$

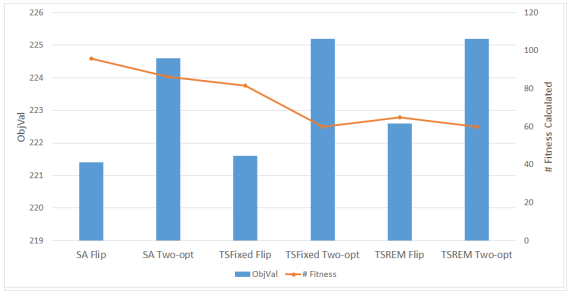


(b) $\theta = 1.4$

Figure 4. Results of $|\mathcal{J}| = 30$.

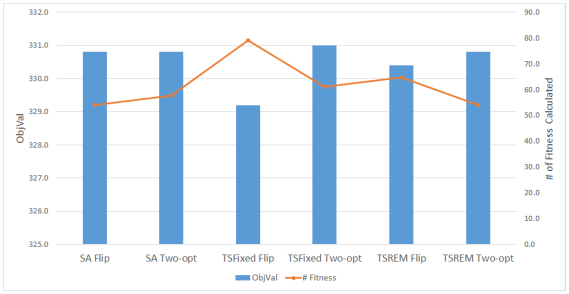


(a) $\theta = 1.2$

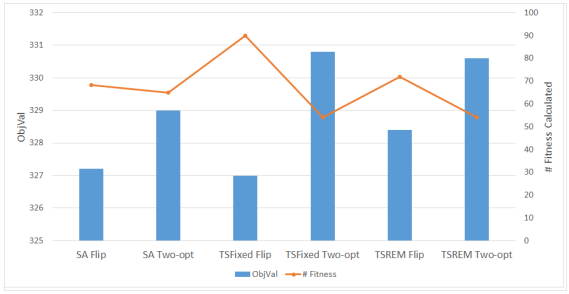


(b) $\theta = 1.4$

Figure 5. Results of $|\mathcal{J}| = 40$.

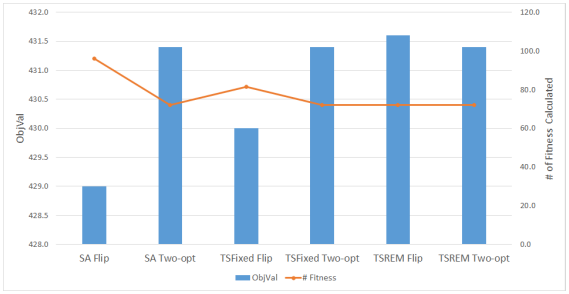


(a) $\theta = 1.2$

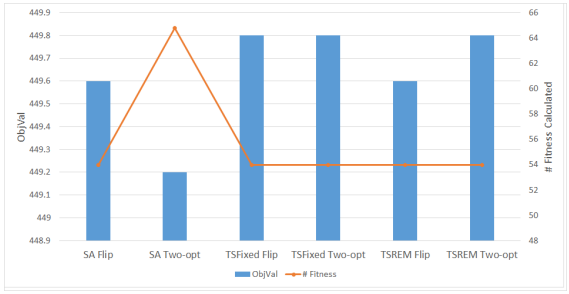


(b) $\theta = 1.4$

Figure 6. Results of $|\mathcal{J}| = 60$.



(a) $\theta = 1.2$



(b) $\theta = 1.4$

Figure 7. Results of $|\mathcal{J}| = 80$.

6. Concluding remarks and future research

This study investigates a two-machine relocation scheduling problem to minimize the makespan. The resource constraints were discussed. The specific setting is about two dedicated machines on the assumptions that each machine is responsible for a given subset of jobs. We strengthen the existing complexity result by proving that the problem remains strongly NP-hard even when (1) the processing sequence on one machine is given and (2) all jobs have positive contributions. This result motivates the two solution approaches of this study: a mixed integer programming model for small instances, and two meta-heuristic algorithms that embed the fixed-sequence dynamic programming algorithm as a subroutine for general instances. A mixed integer programming model was proposed to describe and solve the studied problem. We proposed two meta-heuristic algorithms, using the polynomial-time dynamic programming algorithm that was previously designed for the setting of two fixed sequences, to provide sub-optimal schedules. Computational experiments were conducted to measure the performances of the meta-heuristics as well as the integer programming model.

For future studies, we can consider different objectives, the total completion time $\sum C_{i,j}$, and the total tardiness. To consider such objectives is reasonable because the government responsible for the house redevelopment projects may have other concerns, such as balancing the average waiting time of new buildings and all milestones can be completed on time. For the total tardiness problem with two fixed sequences, the complexity status remains open. Another direction is to extend the RPD2 setting into $RPDm$, where $m \geq 3$ is the number of machines.

Data availability

The test data used in the computational study will be available from the authors upon requests.

Author contributions

Hong: Writing – original draft, Software, Methodology, Investigation, Formal analysis; Wu: Formal analysis, Validation, Writing – review & editing; Lin: Writing – review & editing, Project administration, Methodology, Funding acquisition, Conceptualization.

Use of Generative-AI tools declaration

The authors declare that the preparation of this paper do not use AI tools except for using Gemini to polish the Abstract and the first two paragraphs of Introduction.

Acknowledgments

The authors are grateful to the reviewers for their constructive comments that have greatly improve the presentation and clarified theoretical properties. Part of Lin's work was partially supported by the National Science and Technology Council of Taiwan under grant NSTC114-2410-H-A49-063-MY2.

Conflict of interest

Bertrand M.T. Lin is an editorial board member for Journal of Industrial and Management Optimization and was not involved in the editorial review or the decision to publish this article.

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. C. Artigues, S. Demassey, E. Néron, *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*, London: ISTE Ltd and John Wiley & Sons, 2008.
2. P. Brucker, A. Drexl, R. Möhring, K. Neumann, E. Pesch, Resource-constrained project scheduling: Notation, classification, models, and methods, *Eur. J. Oper. Res.* **112** (1999), 3–41. https://doi.org/10.1016/S0377-2217(98)00204-5
3. S. Hartmann, D. Briskorn, A survey of variants and extensions of the resource-constrained project scheduling problem, *Eur. J. Oper. Res.,* **207** (2010), 1–14. https://doi.org/10.1016/j.ejor.2009.11.005
4. W. Herroelen, B. De Reyck, E. Demeulemeester, Resource-constrained project scheduling: A survey of recent developments, *Comput. Oper. Res.,* **25** (1998), 279–302. https://doi.org/10.1016/S0305-0548(97)00055-5
5. S. Issa, Y. Tu, A survey in the resource-constrained project and multi-project scheduling problems, *J. Proj. Manag.,* **5** (2020), 117–138. https://doi.org/10.5267/j.jpm.2019.11.001
6. L. Özdamar, G. Ulusoy, A survey on the resource-constrained project scheduling problem, *IIE Trans.,* **27** (1995), 574–586. https://doi.org/10.1080/07408179508936773
7. E. H. Kaplan, Relocation models for public housing redevelopment programs, *Environ. Plan. B Plan. Des.,* **13** (1986), 5–19. https://doi.org/10.1177/239980838601300101
8. E. H. Kaplan, A. Amir, A fast feasibility test for relocation problems, *Eur. J. Oper. Res.,* **35** (1988), 201–206. https://doi.org/10.1016/0377-2217(88)90030-6
9. R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey, *Ann. Discrete Math.,* **5** (1979), 287–326. https://doi.org/10.1016/S0167-5060(08)70356-X
10. B. M. T. Lin, F. J. Hwang, A. V. Kononov, Relocation scheduling subject to fixed processing sequences, *J. Sched.,* **19** (2016), 153–163. https://doi.org/10.1007/s10951-015-0455-8
11. S. M. Johnson, Optimal two- and three-stage production schedules with setup times included, *Nav. Res. Logist. Q.,* **1** (1954), 61–68. https://doi.org/10.1002/nav.3800010110
12. E. H. Kaplan, O. Berman, OR hits the heights: Relocation planning at the Orient Heights housing project, *Interfaces,* **18** (1988), 14–22. https://doi.org/10.1287/inte.18.6.14
13. A. Amir, E. H. Kaplan, Relocation problems are hard, *Int. J. Comput. Math.,* **25** (1988), 101–110. https://doi.org/10.1080/00207168808803664

14. A. V. Kononov, B. M. T. Lin, On relocation problems with multiple identical working crews, *Discrete Optim.,* **3** (2006), 366–381. https://doi.org/10.1016/j.disopt.2006.06.003

15. B. M. T. Lin, S. S. Tseng, Some results of relocation problems with processing times and deadlines, *Int. J. Comput. Math.,* **40** (1991), 1–15. https://doi.org/10.1080/00207169108803998

16. B. M. T. Lin, S. S. Tseng, On the relocation problems of maximizing new capacities under a common due-date, *Int. J. Syst. Sci.,* **23** (1992), 1433–1448. https://doi.org/10.1080/00207729208949397

17. B. M. T. Lin, S. T. Liu, Maximizing total reward in the relocation problem subject to generalized due dates, *Int. J. Prod. Econ.,* **115** (2008), 55–63. https://doi.org/10.1016/j.ijpe.2008.04.009

18. N. G. Hall, Scheduling problems with generalized due dates, *IIE Trans.,* **18** (1986), 220–222. https://doi.org/10.1080/07408178608975351

19. S. V. Sevastyanov, B. M. T. Lin, H. L. Huang, Time complexity analysis in the relocation problem with arbitrary release dates, *Theor. Comput. Sci.,* **412** (2011), 4536–4544. https://doi.org/10.1016/j.tcs.2011.04.034

20. Y. C. Su, B. M. T. Lin, Minimizing the total weighted completion time in relocation scheduling, *Comput. Ind. Eng.,* **173** (2022), 108662. https://doi.org/10.1016/j.cie.2022.108662

21. T. C. E. Cheng, B. M. T. Lin, H. L. Huang, Resource-constrained flowshop scheduling with separate resource recycling operations, *Comput. Oper. Res.,* **39** (2012), 1206–1212. https://doi.org/10.1016/j.cor.2010.07.015

22. T. C. Lo, B. M. T. Lin, Relocation scheduling in a two-machine flow shop with resource recycling operations, *Mathematics,* **9** (2021), 1527. https://doi.org/10.3390/math9131527

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

24. S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by simulated annealing, *Science,* **220** (1983), 671–680. https://doi.org/10.1126/science.220.4598.671

25. F. Glover, Future paths for integer programming and links to artificial intelligence, *Comput. Oper. Res.,* **13** (1986), 533–549. https://doi.org/10.1016/0305-0548(86)90048-1

26. F. Glover, Tabu search—Part II, *ORSA J. Comput.,* **2** (1990), 4–32. https://doi.org/10.1287/ijoc.2.1.4

27. F. Glover, Tabu search—Part I, *ORSA J. Comput.,* **1** (1989), 190–206. https://doi.org/10.1287/ijoc.1.3.190

28. F. Dammeyer, S. Voß, Dynamic tabu list management using the reverse elimination method, *Ann. Oper. Res.,* **41** (1993), 29–46. https://doi.org/10.1007/BF02022561 https://doi.org/10.1007/BF02022561

AIMS Press