Exact and Heuristic Methods to Solve the Parallel Machine Scheduling Problem with Multi-processor Tasks

Abstract

This paper studies a special parallel machine scheduling problem where some tasks require more than one machine to process, known as the Parallel Machine Scheduling Problem with Multi-processor Tasks. Two mathematical models and several theoretical properties are proposed for the studied problem. To solve this problem, this paper develops an exact branch and bound algorithm and a heuristic tabu search algorithm. A series of numerical experiments are conducted to test the performance of these solution methods. The computational results show that the solution methods are effective and efficient in solving the problem with different sizes.

Keywords: Scheduling, Parallel Machine, Multi-processor Tasks, Branch and Bound, Tabu Search

1. Introduction

Parallel machine scheduling problems consist of assigning and sequencing a set of tasks on a set of machines. One of the core assumptions made in classical parallel machine scheduling problems is that each task needs to be processed on exactly one machine at a time. However, this assumption is insufficient to handle many problems arising from today's production realities. Namely, tasks in some scheduling problems require more than one machine to process, and we call such problems the *multi-processor task scheduling problems* (Blazewicz et al., 1986). There are a large number of industrial applications where *multi-processor task scheduling problems* occur. For instance, (1) in diagnosable microprocessor systems, a task must be processed by at least two processors at the same time (Krawczyk and Kubale, 1985); (2) in semiconductor circuit design workforce planning, a design project may need a group of people (Chen and Lee, 1999), (3) in the quay crane scheduling problem of container terminals, a ship requires more than one crane to handle (Trkoullar et al., 2014) and (4) in parallel batch jobs scheduling problems of grid computing environments, a computing task may run on several processors that work in parallel (Switalski and Seredynski, 2015).

This paper studies a Parallel Machine Scheduling problem which allows tasks to be processed on more than one machine. The considered objective in our work is to optimize the makespan (the latest completion time of all tasks). In this paper, we use the three-field notation, $\alpha|\beta|\gamma$ which is introduced by Graham et al. (1979) for parallel machine scheduling problems. In this notation: (1) the first field α specifies the machine environment, where P represents identical, Q uniform, and R unrelated parallel machines, (2) the second field β denotes the job characteristics, and the third field γ denotes the objective function. According to Drozdowski (1996)'s classification, this problem can be denoted by $P|size_j|C_{\max}$, where P represents parallel machine, $size_j$ denotes the number of machines required by task j and $C_{\max}$ indicates that the objective is to optimize the makespan.

This paper deals with the $P|size_j|C_{\max}$ problem, and the main contribution includes:

- This paper theoretically analyzes the $P|size_j|C_{\max}$ problem and presents formal formulations. Several theoretical properties along with two different mixed integer programming models of the considered problem are proposed.

- To solve problems with real-world sizes, a branch and bound (B & B) algorithm with five efficient lower bounds and two fathoming criteria is proposed. For problems with even larger scales, we develop a tabu search (TS) method.
- Numerical experiments involving 700 instances are conducted. Computing results indicate that the proposed solution methods can obtain high-quality solutions for the $P|size_j|C_{\max}$ problem with different sizes in relatively short computational time.

This paper is organized as follows. The next section gives a review of related studies. In Section 3, we formally state the $P|size_j|C_{\max}$ problem and present several properties for the optimal schedule of the problem. Two different mixed integer programming models for the problem are formulated in Section 4. A branch and bound method is proposed in Section 5. In Section 6 we introduce a tabu search algorithm. The performances of the proposed models and algorithms are examined in Section 7 by a series of computational experiments. Finally, our findings are summarized in Section 8.

2. Literature review

Studies of the $P|size_j|C_{\max}$ problem were motivated by research in computer systems, such as fault-tolerant systems (Krawczyk and Kubale, 1985). Lloyd (1981) is among the first who studied the $P|size_j|C_{\max}$ problem. In his paper the problem in which all tasks share equal processing time, $P|size_j, p_j = 1|C_{\max}$, where p_j represents processing time is studied and it was shown that the general $P|size_j, p_j = 1|C_{\max}$ problem is NP-complete. The author also demonstrated that the performance bound of list scheduling for solving the $P|size_j, p_j = 1|C_{\max}$ problem is bounded by $(2m - k)/(m - k + 1)$, where m is the number of machines and k is the largest number of machines required by one task. Blazewicz et al. (1986) considered the preemptive and nonpreemptive parallel machine scheduling problems with multiprocessor tasks. They proposed two polynomial-time algorithms with the complexity of $O(n)$ to two problems where each task requires either 1 or k machines to process ($size_j \in \{1, k\}$, where k is integer and $k \leq m$). One problem is the $P|size_j \in \{1, k\}, p_j = 1|C_{\max}$ problem and the other can be denoted by the $P|size_j \in \{1, k\}, pmtn|C_{\max}$ problem in which $pmtn$ implies that tasks are preemptive. Computational complexity of the $P|size_j|C_{\max}$ problem was examined by Du and Leung (1989). Their study showed that for $m = 2, 3$, the $P|size_j|C_{\max}$ problem can be solved in pseudopolynomial time but the problem becomes NP-hard in the strong sense for each $m \geq 5$. Blkadek et al. (2015) studied the differences between contiguous schedules and non-continuous schedules of the $P|size_j|C_{\max}$. In contiguous schedules, indices of the processors assigned to a task must be a sequence of consecutive numbers while in non-continuous schedules indices of the processors assigned to a task can be arbitrary. They proved that deciding whether such difference exists for a $P|size_j|C_{\max}$ instance is NP-complete and provided bounds on the difference between the length of two schedules. Besides the parallel machine setting, there are some studies that tackle machine scheduling problem with multi-processor tasks in the flow shop environment (e.g., Chou (2013); Hidri (2016)).

As for the solution methods of the $P|size_j|C_{\max}$ problem, exact solution methods can be found for some special cases of the problem, for example, when all the tasks share the same processing time or there are only two different types of machine requirements (Blazewicz et al., 1986), when tasks have various ready times and require either one or all machines (Blazewicz et al., 2003), or when only two or three machines are involved (Du and Leung, 1989; Chen and Lee, 1999). When the number of machines is fixed and not a parameter of the problem, Amoura et al. (2002) proposed a polynomial time approximation scheme for the problem $Pm|size_j|C_{\max}$, where Pm means that the number of identical machines (m) is fixed. When it comes to the general $P|size_j|C_{\max}$ problem, to the best of our knowledge, the only exact method was proposed by Blkadek et al. (2015). The authors developed a simple branch and bound (B & B) algorithm to solve the $P|size_j|C_{\max}$. The algorithm searches the solution space by enumerating all possible permutations of tasks. Partial

permutations of tasks which represents sequences of completed tasks are expanded step by step adding undone tasks. For each newly generated partial solution, only one simple lower bound (the makespan of the partial solution) is used in the algorithm to determine the validity of the partial solution. Using a cluster of 30 PCs, the algorithm was reported to be able to deliver optimal solutions for instances with up to 11 tasks. Several approximation methods have been proposed to solve the problem. Lin and Chen (1994) obtained a heuristic algorithm based on the well-known *largest processing time first scheduling, LPT* rule. The performance bound of this algorithm is proved to be $\frac{4}{3}k - \frac{k(k+1)}{6m}$, where k equals the largest number of machines required by each task. A similar approximation method for the $P|size_j|C_{\max}$ problem was developed by Li (1999). The method was shown to have a performance guarantee of $\frac{31}{18}$. Later on, Johannes (2006) proved that no approximation algorithm for the $P|size_j|C_{\max}$ problem can provide with a performance guarantee better than $\frac{3}{2}$, unless P = NP. The study also demonstrated the performance guarantee of any list-scheduling algorithm for the $P|size_j|C_{\max}$ problem should be no less than 2. Switalski and Seredynski (2015) proposed an evolutionary metaheuristic algorithm called the generalized extremal optimization (GEO) to solve a machine scheduling problem with multi-processor tasks. In their study, each machine is defined to have a batch of processors that can work in parallel, and different machines may have different numbers of processors. Tasks with different processor requirements are scheduled among these machines (each task can only be assigned to one machine). Instances with up to 500 tasks and 48 machines are solved and compared with the genetic algorithm (GA). The computational results showed that the proposed GEO outperforms the GA for most of the tested instances.

For a comprehensive understanding of this problem, refer to the studies of Drozdowski (1996), Lee et al. (1997), Brucker (2006) and Leung (2004).

The $P|size_j|C_{\max}$ problem has wide-range industrial applications and has been studied by a number of scholars. In this study, we analyze several properties of the optimal solution of the $P|size_j|C_{\max}$ problem, formulate two different programming models and propose an exact branch and bound (B & B) method and a tabu search (TS) heuristic to solve the problem. Numerical experiments show that the proposed algorithms are effective and efficient to solve the $P|size_j|C_{\max}$ problem with different sizes.

3. Problem description

This section first gives a formal description of the parallel machine scheduling problem with multi-processor tasks, and then presents several properties of an optimal schedule to this problem.

This problem can be described as follows: assume there is a set $N = \{1, 2, ..., n\}$ of n tasks to be processed by the set $M = \{1, 2, ..., m\}$ of m identical machines. All the tasks are available and all the machines are ready to work from time $t = 0$. Each machine can process at most one task at a time, and each task j needs to be processed by $size_j$ arbitrary machines working in parallel ($size_j$ is integer and $1 \leq size_j \leq m$) in a continuous period of p_j units of time (p_j is integer and $p_j \geq 1$). Preemption is not allowed. The objective is to find a schedule that produces the minimum makespan ($C_{\max}$).

The optimal schedule of the $P|size_j|C_{\max}$ problem has several properties, which can help decrease the search space for the solution algorithms. The first property examines the effectiveness of List Scheduling (LS) rule for the considered problem. List Scheduling rule assigns the earliest available machines to each task in a given sequence.

Theorem 3.1. *There exists an optimal schedule for the $P|size_j|C_{\max}$ problem where machines are assigned to tasks by the List Scheduling rule.*

Proof. Let δ denote a schedule of all tasks in our considered problem. In addition, we assume in schedule δ, $C(\delta)_j$ is the completion time of task j and $C_{\max}(\delta)$ is the makespan. Since

$C_{\max}(\delta_1) > C_{\max}(\delta_2)$ implies $C(\delta_1)_j > C(\delta_2)_j$ for at least one j, the objective function $C_{\max}(\delta)$ is a regular function of δ. Further, for any scheduling problem with a regular objective function, there exists an optimal schedule that is generated by the List Scheduling rule (Schutten, 1996).□

The second property specifies the sequence of certain tasks in an optimal schedule. Assume $\hat{s} = \min_{j \in N} size_j$, then we have the following theorem:

Theorem 3.2. *There exists an optimal schedule for the $P|size_j|C_{\max}$ problem where tasks with $size_j > m - \hat{s}$ are processed in arbitrary sequences before all the other tasks start.*

Proof. We prove by contradiction. Let $\theta = \{j | size_j > m - \hat{s}\}$ and let Δ denote the set of all schedules to the problem $P|size_j|C_{\max}$ and Δ_s denote the set of schedules in which all tasks belonging to θ are processed before all the other tasks. Suppose there exists an optimal schedule $\bar{\delta} \in \Delta \setminus \Delta_s$ such that for all $\delta \in \Delta_s$, $C_{\max}(\bar{\delta}) < C_{\max}(\delta)$. Let $\theta^{'}$ denote a subset of θ in which tasks are processed after a set of tasks belonging to $N \setminus \theta$ are completed. Note that due to the limitation of the total number of machines available, no tasks can be processed simultaneously when a task belonging to θ is being processed.

Suppose there is a task $u \in \theta^{'}$. We now split $\bar{\delta}$ into several parts. Assume δ_1 is a partial schedule of $\bar{\delta}$ composed of tasks which belong to θ and are processed before any task from $N \setminus \theta$ starts. Further, assume δ_2 is a partial schedule which contains tasks that are processed after tasks in δ_1 are completed and before task u starts. Finally, the partial schedule including tasks processed after task u is completed is denoted by δ_3. It is worth noting that since no two tasks from θ can be processed simultaneously, the makespan of δ_1 equals the summation of the processing time of all tasks belonging to it.

Let $MC(\delta)$ denote the set of completion times of machines in the schedule δ, and $MINC_k(\delta)$ denote the kth minimum value in the set $MC(\delta)$. As no tasks can be processed simultaneously with task u, the earliest starting time of tasks belonging to δ_3 is $MINC_m(\delta_2) + p_u$.

We now construct a new schedule $\bar{\delta}^{'}$ of the problem $P|size_j|C_{\max}$ by adding task u into δ_1. It is obvious that such an adding will not change the sequences of tasks in δ_2 and δ_3, but the schedules of these tasks may be changed. We denote the schedules of tasks from δ_1, δ_2, and δ_3 in the new schedule as $\delta_1^{'}$, $\delta_2^{'}$ and $\delta_3^{'}$, respectively. Since no tasks can be processed at the same time with task u, it is easy to infer that $MINC_i(\delta_2^{'}) = MINC_i(\delta_2) + p_u$. This means that tasks in $\delta_3^{'}$ can start at $MINC_k(\delta_2^{'})$, which equals $MINC_k(\delta_2) + p_u$ where k represents the number of machines required by the first task in $\delta_3^{'}$. As $k \leq m$, it can be easily inferred that $MINC_k(\delta_2) + p_u \leq MINC_m(\delta_2) + p_u$. This implies that $C_{\max}(\bar{\delta}^{'}) \leq C_{\max}(\bar{\delta})$.

If we apply the above procedures to all the tasks belonging to $\theta^{'}$, we can obtain a schedule belonging to Δ_s that is at least as good as $\bar{\delta}$, which is a contradiction of the minimality of $C_{\max}(\delta^{'})$.□

Corollary 3.1. *When $m = 2$, the problem $P2|size_j|C_{\max}$ can be solved as a $P2||C_{\max}$ problem, where each task needs to be processed on only one machine.*

Proof. The corollary follows directly from Theorem 3.2.□

4. Model formulation

Although the $P|size_j|C_{\max}$ problem has been studied by the literature for more than 30 years, to the best of our knowledge, there have been yet no formal mathematical optimization formulations reported for this problem. In this section, we present two mixed integer programming models for the $P|size_j|C_{\max}$ problem. In the first model, the problem is formulated as a special *Vehicle Routing Problem (VRP)* and in the second model, a time discretized formulation is developed.

4.1. A VRP-based model

The first model (M1) formulates the $P|size_j|C_{\max}$ problem as a *Vehicle Routing Problem with side constraints*. In the model, machines are defined as vehicles and tasks are defined as nodes that must be visited by a certain number of vehicles at the same time. Besides, the traveling time between a node and other nodes is assumed to be equal to the processing time of the corresponding tasks.

Besides the parameters that have been listed in the previous section, additional notations used in Model M1 are presented in Table 1:

Table 1: Notations used in Model M1.

Parameters:	
i, j	Indices of tasks, $i \neq j$.
k:	Index of machines.
L:	A constant number, $L = \sum_{j \in N} p_j$.
Variables:	
x_{ij}^k	= 1, if machine k performs task j before any other tasks after it completes task i; =0, otherwise.
x_{0j}^k	= 1, if the first task performed by machine k is j; = 0, otherwise.
x_{iT}^k	= 1, if the last task performed by machine k is i; = 0, otherwise.
s_i	Start time of task i.
Y_k	Completion time of machine k.

The first formulation (M1) of the $P|size_j|C_{\max}$ problem goes as follows:
Objective function:

$$\min C_{\max} \tag{1}$$

subject to:

$$Y_k \leq C_{\max} \quad \forall k \in M \tag{2}$$

$$\sum_{j \in N} x_{0j}^k \leq 1 \quad \forall k \in M \tag{3}$$

$$\sum_{i \in N} x_{iT}^k \leq 1 \quad \forall k \in M \tag{4}$$

$$\sum_{k \in M} \sum_{i \in N \cup 0} x_{ij}^k = size_j \quad \forall j \in N \tag{5}$$

$$\sum_{j \in N \bigcup T} x_{ij}^k - \sum_{j \in N \bigcup 0} x_{ji}^k = 0 \quad \forall i \in N, \forall k \in M \tag{6}$$

$$s_i - s_j + p_i \leq (1 - x_{ij}^k)L \quad \forall i, j \in N, \forall k \in M \tag{7}$$

$$s_i + p_i - Y_k \leq (1 - x_{iT}^k)L \quad \forall i \in N, \forall k \in M \tag{8}$$

$$x_{ij}^k \in \{0, 1\} \quad \forall i \in N \cup 0, \forall j \in N \cup T, \forall k \in M \tag{9}$$

$$s_i \geq 0 \quad \forall i \in \Omega \tag{10}$$

The objective function (1) minimizes the makespan of the problem. Constraint (2) calculates the makespan. Constraint (3) enforces that every machine chooses at most one task as its first task. Constraint (4) enforces that every machine chooses at most one task as its last task. Constraint (5)

ensures that each task should be processed by the required number of machines. Constraint (6) ensures flow balance so that all tasks are performed in a defined order. Constraint (7) ensures the start time of task j should be later than the completion time of task i, if $x_{ij}^k = 1$. Constraint (8) calculates the time when machine k completes all of its assigned tasks. Constraints (9) defines the x_{ij}^k variables to be binary. Constraint (10) defines the non-negativity constraint.

It should be noted that although the $P|size_j|C_{\max}$ problem can be formulated as a special VRP and there are many state-of-the-art exact and heuristic algorithms in the literature for solving the VRP, these algorithms cannot be easily adapted to the considered problem. This is because (1) in classical VRPs, the service of each customer (task) requires exactly one vehicle (machine) and (2) the objective for classical VRPs is to minimize total traveling cost but not the makespan.

4.2. A time-discretized model

In the second model (M2), a set of time-discretized 0-1 decision variables x_{jt}, where $j \in N$ and $t \in T$ are introduced. The set T is defined as $T = \{0, 1, 2, ..., t_{\max}\}$, where $t_{\max}$ denotes the largest possible makespan for an optimal schedule which is calculated by $t_{\max} = \sum_{j \in N} p_j$. Further, we set $E_j = t_{\max} - p_j$ for each $j \in N$, representing the latest start time of task j in an optimal schedule. For each task j and time t, the x_{jt} is defined as 1, if task j starts at time t and 0, otherwise.

The second model (M2) for the considered problem is given as follows:

Objective function:

$$\min C_{\max} \tag{11}$$

subject to:

$$\sum_{t=0}^{E_j} x_{jt} = 1 \quad \forall j \in N \tag{12}$$

$$x_{jt}(t + p_j) \le C_{\max} \quad \forall j \in N, \forall t \in T \tag{13}$$

$$\sum_{j \in N} \sum_{s=\max(t-p_j+1,0)}^{\min(t,E_j)} size_j x_{js} \le m \quad \forall t \in T \tag{14}$$

$$x_{jt} \in \{0, 1\} \quad \forall j \in N, \forall t \in T \tag{15}$$

Constraint (12) ensures that each task has one and only one start time. Constraint (13) calculates the makespan. Constraint (14) enforces the machine availability constraint. Constraint (15) defines the variables x_{jt} to be binary. In addition, it is worth noting that the solution of M2 does not directly give the optimal machine assignment pattern, which, however, can be obtained using the LS rule in $O(nm \log_2 m)$ time.

In Section 7, we test the efficiencies of the two models by solving a series of numerical experiments, and further compare performances of CPLEX using the two models with the performances of the algorithms proposed in the next two sections for solving these instances.

5. A branch and bound method

The $P|size_j|C_{\max}$ problem is NP-hard for any $m \ge 2$ (Blazewicz et al., 1983) and is proved to be NP-hard in the strong sense for at least any $m \ge 5$ (Du and Leung, 1989). Correspondingly, the $P|size_j|C_{\max}$ problem cannot be solved exactly in polynomial time for any $m \ge 2$ and even cannot be solved exactly in pseudo-polynomial time at least when $m \ge 5$, unless P=NP. Hence, to solve the $P|size_j|C_{\max}$ problem efficiently, this paper proposes a B & B method.

In the B & B method, solutions of the $P|size_j|C_{\max}$ problem are represented by sequences of tasks. The assignments of machines to each task in the sequences are generated by the LS rule, which, according to Theorem 3.1, is able to generate an optimal solution for the considered problem. The branch and bound procedure builds up a solution tree starting from a root node representing level 0. The number of levels of the solution tree is the same as the number of tasks, and the node in the ith level representing the ith task in a corresponding task sequence. Obviously, each path from a node in the first level to a node in the ith level corresponds to a partial solution of the problem consisting of i tasks. The flow of the B & B method is shown in Figure 1.

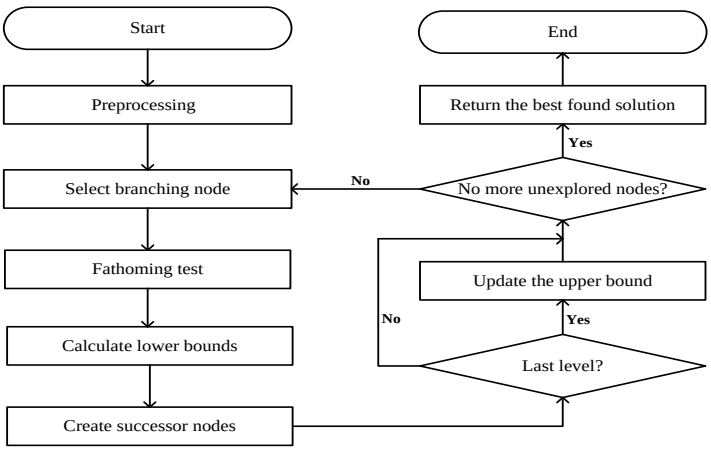

Figure 1: The flowchart of the B & B method.

The notations used in this section are as followings: For a certain node h in the search tree, $D(h)$ and $U(h)$ are defined as the sets of completed and uncompleted tasks at the node, respectively. Let $G(h)$ denote an array which is composed of the completion times of all machines at the node, in a chronological order, that is $G(h) = [g_1^h, g_2^h, \cdots, g_k^h, g_{k+1}^h, \cdots, g_m^h]$, $g_k^h \leq g_{k+1}^h$, $\forall k = 1, 2, \cdots, n-1$. Further, we divide tasks in N, $D(h)$, and $U(h)$ into different subsets N_k, $U(h)_k$ and $D(h)_k$, where k stands for the number of machines they require. Namely, $N = \bigcup_{k \in M} N_k$, $D(h) = \bigcup_{k \in M} D(h)_k$ and $U(h) = \bigcup_{k \in M} U(h)_k$. Obviously, for any node h, we have $N_k = D(h)_k \bigcup U(h)_k$, $\forall k \in M$.

5.1. Preprocessing

Before the B & B method starts, we reduce the size of the search tree of the B & B algorithm by removing tasks with $size_j > m - \hat{s}$ from N, where $\hat{s} = \min_{j \in N} size_j$. These tasks, according to Theorem 3.2, can be processed in an arbitrary sequence before any other tasks start in an optimal schedule. However, it is worth noting that the derived makespan of the tasks with $size_j \leq m - \hat{s}$ by the B & B method should be added by the total processing time of the removed tasks to obtain the makespan of a complete schedule. For conciseness, tasks in the rest part of this section only refer to tasks with $size_j \leq m - \hat{s}$.

5.2. Branching

To continue the partial task sequence, selected nodes at level i are branched by adding successor nodes to the search tree at level $i + 1$.

Two different selection strategies are adopted in the B & B method. The first strategy selects the candidate node for branching in a *depth-first* manner. The branching node is selected from the highest level with unexplored nodes and the selection criterion is the minimum lower bound (see Section 5.3.2 for details). In the second strategy, the unexplored node with the overall minimum lower bound throughout the search tree is selected as the candidate for branching. We implement

the two strategies in an alternative way so as to strike a balance between the speed of finding new solutions and the quality of these new solutions, both of which are important for finding tighter upper bounds. After the selection of branching nodes at level i, the successor nodes are generated at level $i + 1$ by enumerating and exploring all the remaining tasks.

5.2.1. Fathoming test

To improve the efficiency of the searching process, the size of the search tree is reduced by fathoming nodes that are dominated by others. Assume there are two nodes h and d in the search tree. If node d is expected to be able to produce a makespan that is no larger than the makespan produced by node h, then node h is judged to be dominated by node d. We introduce three dominance properties for the fathoming test.

The first and second properties check possible dominance within one partial schedule. For each task j in $D(h)$, assume Ω_j is the set of $size_j$ machines that are assigned for j. Let BIT_j^k denote the kth longest idle time of a machine in Ω_j before it is assigned to task j. Furthermore, let AIT_j denote the minimum idle time of all machines in Ω_j after they complete task j and before they are reassigned to other tasks in $D(h)$, then we have:

Property 5.1. *(Blkadek et al., 2015) For node h, if there exists a task j in D(h) such that a task i in U(h) satisfies $p_i \leq BIT_j^k$, where $k = size_i$, then node h is dominated by some other node.*

Property 5.2. *For node h, if there exists a task j in D(h) such that a task i in N_s where $s = size_j$ satisfies $p_j < p_i \leq p_j + AIL_j$, then node h is dominated by some other node.*

Proof. Let δ represent the corresponding partial schedule at node h. We construct a new partial schedule δ' by interchanging tasks j and i, and the $U(h)$ and $G(h)$ for δ are correspondingly changed into $U(h)'$ and $G(h)'$ in δ'. We distinguish two cases.

Case 1. $i \in U(h)_s$.

In this case, δ' is constructed by replacing task j with task i in $U(h)$, and $U(h)'$ is obtained by replacing i with j in $U(h)$. We now prove δ is dominated by δ'. As $p_j < p_i \leq p_j + AIT_j$ and $size_i = size_j$, task i can be processed by the same machines belonging to Ω_j. Hence, δ' can be generated by replacing task j with task i in δ and leaving the schedules for other tasks in $D(h)$ unchanged. Further, since all machines in Ω_j have been reassigned in δ, j is not the last processed task for any machine in δ. Therefore, we have $G(h)' = G(h)$, which implies that ready times of all machines for tasks in $U(h)$ and those in $U(h)'$ are identical.

We further compare the sets of uncompleted tasks $U(h)$ and $U(h)'$. The only difference between these two sets is that in $U(h)'$, task j in $U(h)$ is replaced with task i. Since $p_j < p_i$, $size_j = size_i$ and $G(h)' = G(h)$, it is easy to infer that the makespan of the optimal complete schedule derived from δ' is no larger than the makepan of the optimal complete schedule derived from δ. Therefore, δ is dominated by δ', which means that node h is dominated by some other node.

Case 2. $i \in D(h)_s$.

In this case, we construct a new partial schedule δ' by interchanging the processing sequences of task j and i. As $size_j$=$size_i$, $p_i \leq p_j + AIT_j$ and $p_j < p_i$, it is easy to infer that tasks which are processed by at least one of the machines of Ω_j immediately after j is completed in δ can start in δ' no later than in δ, and similarly, that tasks which are processed by at least one of the machines of Ω_i immediately after i is completed in δ can start in δ' no later than in δ.

Hence, for each $k \in M$ we have $g_k^{h'} \in G(h)'$ is no larger than $g_k^h \in G(h)$. Further, considering the changes in δ have no impact on $U(h)$ ($U(h) = U(h)'$), we can obtain that the makespan of the

optimal complete schedule derived from δ' is no larger than the makepan of the optimal complete schedule derived from δ. Therefore, δ is dominated by δ', which means that node h is dominated by some other node.□

The second property detects the dominating relationship between two nodes in the same level of a search tree.

Property 5.3. *Suppose there are two nodes h and d with $U(h) = U(d)$, then node h is considered to be dominated by node d if the $g_k^d \leq g_k^h$ for each $k \geq S_u$ and $k \leq m$, where $U = U(h) = U(d)$ and $S_u = \min_{j \in U} size_j$.*

Proof. Given that each task in $U(h)$ and $U(d)$ needs at least S_u machines to process, the least possible ready times for all tasks belonging to $U(h)$ and $U(d)$ are $g_{S_u}^h$ and $g_{S_u}^d$, respectively. In this sense, we can take $g_k^h = g_{S_u}^h$ and $g_k^d = g_{S_u}^d$ for any $k < S_u$ in $G(h)$ and $G(d)$. This implies that $g_k^d \leq g_k^h$ for each $k \in M$.

In addition, as the two nodes h and d share the same set of uncompleted tasks, it is not hard to infer that node d can produce an optimal schedule whose makespan is at least as short as that of the optimal schedule produced by node h. Therefore, h is dominated by node d.□

The proposed B & B checks possible dominance relationship of every newly added node according to Property 5.1, 5.2 and 5.3. The nodes that are found to be dominated by others are fathomed, along with the nodes generated from them (if applicable).

5.2.2. Update the upper bound

The insertion of any node in the last level corresponds to a new solution of the problem. Objective values of these new solutions are compared with the current upper bound to determine whether an update is needed.

5.3. Bounding

For each newly added node, a lower bound is calculated which is then compared with the upper bound to decide whether this node should be retained or fathomed. In the algorithm, we consider two initial upper bounds and five lower bounds.

5.3.1. Initial upper bound

To get the initial upper bound for our B & B procedure, this paper adopts two approximation algorithms to solve the $P|size_j|C_{\max}$ problem. The first algorithm processes tasks with largest $size_j \cdot p_j$ first and assigns machines by the LS rule, while the second algorithm is applied according to the descriptions in Li (1999). The initial upper bound is set equal to the makespan derived from the approximation algorithm that generates the better solution.

5.3.2. Lower bounds

At each node (marked as node h) in the search tree, there exists a partial scheduling problem which consists of a set of machines with ready times from $G(h)$ and a set of tasks belonging to $U(h)$. The problem can be denoted by $P|size_j, r_k|C_{\max}$, where r_k stands for the ready time of machine k. We further introduce two problems which are related to the $P|size_j, r_k|C_{\max}$ problem but some conditions are relaxed. In the first problem, we assume tasks are "divisible", which means a task j requiring $size_j$ machines to process during p_j time is divided into $size_j$ tasks each of which needs only one machine to process during p_j time. Such a problem can be seen as the parallel machine scheduling problem in the traditional sense and we denote the problem by $P|r_k|C_{\max}$. The second problem is denoted by $P|prmt, r_k|C_{\max}$ where tasks are not only "divisible" but also preemptive. Obviously, among the three problems, the $P|prmt, r_k|C_{\max}$ problem allows the most

balanced distribution of task processing time among machines, followed by the $P|r_k|C_{\max}$ problem and then the $P|size_j, r_k|C_{\max}$ problem. Therefore, the $P|prmt, r_k|C_{\max}$ problem and the $P|r_k|C_{\max}$ problem should produce shorter makespans than the $P|size_j, r_k|C_{\max}$ problem, and correspondingly, lower bounds to the first two problems should also be the lower bounds to the $P|size_j, r_k|C_{\max}$ problem. In addition, function $\max[q]_{j\in\Theta}x_j$ which returns the qth largest value of x_j for all $j \in \Theta$ is also used in lower bound calculation.

Lower bound 1:

We obtain the first lower bound by solving the corresponding $P|prmt, r_k|C_{\max}$ problem, and the lower bound for node h is calculated by:

$$LB_1 = \begin{cases} g_m^h & UT \leq DM \\ g_m^h + \text{Ceil}((UT - DM)/m) & UT > DM \end{cases} \tag{16}$$

where $UT = \sum_{j\in U(h)} size_j p_j$ and $DM = \sum_{k\in M}(g_m^h - g_k^h)$.

Lower bound 2:

The Lower bound 2 for note h is obtained in a multi-lift procedure. Assume μ =Floor$(m/2)$, and Lower bound 2 can be calculated by the following equation:

$$LB_2 = \max_{k\in\{1,2,...,\mu\}\bigcup\{m\}} CT_k \tag{17}$$

where CT_k represents the lower bound for the completion time of tasks belonging to $U(h)_k$ if $k \in \{1, 2, ..., \mu\}$ and the lower bound for the completion time of tasks belonging to $\bigcup_{k\in\{\mu+1,\mu+2,...,m\}} U(h)_k$ if $k = m$.

As for the estimation of CT_ks, we first discuss the situation where $k \in \{1, 2, ..., \mu\}$. For each k, CT_k is given by:

$$CT_k = \max(CT_k^1, CT_k^2) \tag{18}$$

where CT_k^1 is calculated as:

$$CT_k^1 = g_k^h + \max_{j\in U(h)_k} p_j, \forall k \in \{1, 2, ..., \mu\} \tag{19}$$

The calculation of CT_k^2 is more complicated. For each CT_k^2, a set $V(h)_k$ containing l new tasks is generated, where l =Floor(m/k). These tasks require k machines and the processing time for task j $(1 \leq j \leq l)$ in $V(h)_k$ is set as $g_{i\cdot k}^h - g_1^h$. For each k, CT_k^2 is obtained by the following equation:

$$CT_k^2 = g_1^h + \max[l]_{j\in V(h)_k \bigcup U(h)_k} p_j + \max[l+1]_{j\in V(h)_k \bigcup U(h)_k} p_j \tag{20}$$

We now give justifications for CT_k^2. We first construct two partial scheduling problems P1 and P2 at node h. P1 includes a set of m machines with ready times from $G(h)$ and tasks belonging to $U(h)_k$, and P2 covers a set of m machines whose ready times are all g_1^h and a set tasks belonging to $V(h)_k \bigcup U(h)_k$. Clearly, CT_k^2 is a lower bound for P2. Further, P1 can be seen as a special partial solution to P2, where tasks from $V(h)_k$ are processed prior to those from $U(h)_k$. Therefore, the optimal makespan of P2 is obviously no larger than that of P1, and lower bounds to P2 should also be lower bounds to P1. Considering P1 is also only part of the $P|size_j, r_k|C_{\max}$ problem, CT_k^2 is a valid lower bound for node h as well.

When $k = m$, the CT_m stands for the lower bound for the completion time of tasks belonging to $\bigcup_{k\in\{\mu+1,\mu+2,...,m\}} U(h)_k$. Because of their machine requirements, no two tasks of this union set can

be processed at the same time. Hence, the CT_m can be obtained by:

$$CT_m = \max_{k=\{\mu+1,\mu+2,\ldots,m\}} (g_k^h + \sum_{b=k}^{m} \sum_{j \in U(h)_b} p_j) \tag{21}$$

Lower bound 3:

The third lower bound of the B & B method for node h is obtained by the following equation:

$$LB_3 = \max(LB_3^1, LB_3^2) \tag{22}$$

LB_3^1 is calculated by treating the partial scheduling problem with machine ready times belonging to $G(h)$ and tasks belonging to $U(h)$ as a corresponding $P|r_k|C_{\max}$ problem, and generating a lower bound for the $P|r_k|C_{\max}$ problem. We construct a set $V(h)$ of $m-1$ tasks where task j $(2 \leq j \leq m)$ needs one machine to process in $g_j^h - g_1^h$ time. Besides, every task j in $U(h)$ is divided into $size_j$ tasks, each of which requires one machine to process during p_j time and these divided tasks form a new set $U(h)^{'}$.

Then, LB_3^1 is given by:

$$LB_3^1 = g_1^h + \max[m]_{j \in V(h) \bigcup U(h)^{'}} p_j + \max[m+1]_{j \in V(h) \bigcup U(h)^{'}} p_j \tag{23}$$

Like the CT_k^2 in Lower bound 2, LB_3^1 is also obtained by taking part of the $G(h)$ as tasks and therefore, it is not hard to infer that LB_3^1 is a lower bound for the $P|r_k|C_{\max}$ problem and therefore, also a lower bound for the $P|size_j, r_k|C_{\max}$ problem at node h.

To calculate LB_3^2, we generate $m-1$ new tasks which are marked as t_i $(1 \leq i \leq m-1)$ based on $G(h)$. The number of machines needed by t_i is set to be $m-i$ and the processing time is set to be $g_{(i+1)}^h - g_i^h$. A set $U(h)^{'}$ is constructed by adding the $m-1$ tasks into $U(h)$. The calculation of LB_3^2 is completed in 6 steps:

Step 1: Set the occupied machine number $om = 0$ and machine working time $w_k = 0, \forall k \in M$;

Step 2: If $om < m$, go to step 3; otherwise, go to step 6;

Step 3: Identify the task in $U(h)^{'}$ with $p_j = \max_{j \in U(h)^{'}} p_j$ and mark the task as J.

Step 4: If $om + size_J \leq m$, update $w_k = p_J$ for $k = om+1, om+2, ..., om+size_J$, and then set $om = om + size_J$; otherwise, go to step 5, and delete task J from $U(h)^{'}$;

Step 5: update $w_k = p_J$ for $k = om+1, om+2, ..., m$, add a new task i with $p_i = p_J$ and $size_i = size_J - m + om$ into $U(h)^{'}$, update $om = m$, and delete task J from $U(h)^{'}$;

Step 6: Calculate LB_3^2 by the following equation:

$$LB_3^2 = g_1^h + \max_{j \in U(h)^{'}} (w_{m-size_j+1} + p_j) \tag{24}$$

Considering LB_3^2 is obtained through taking part of the $G(h)$ as tasks and in some cases dividing tasks, we can easily infer that it is a valid lower bound of node h.

Lower bound 4:

Like LB_2, the fourth lower bound is also obtained in a multi-lift procedure:

$$LB_4 = g_1^h + \max_{e\in\{\mu+1,\mu+2,...,m\}}(LB_4^e) \tag{25}$$

$$LB_4^e = L_e + M_e \tag{26}$$

where μ =Floor$(m/2)$ and L_e and M_e are explained below.

First, part of the $G(h)$ is transformed into tasks. For each $k \in \{2, 3, ..., m\}$, we generate a task j which requires $m - k + 1$ machines to process during $g_k^h - g_{k-1}^h$ time and introduce a set $U(h)'_{m-k+1} = U(h)_{m-k+1} \bigcup \{j\}$. Then, for each $e \in \mu + 1, \mu + 2, ..., m$, the following three union sets are constructed: $U_L^e = \bigcup_{k\in\{e,e+1,...,m\}} U(h)'_k$ and $U_M^e = \bigcup_{k\in\{m-e+1,m-e+2,...,e-1\}} U(h)'_k$, and $U_S^e = \bigcup_{k\in\{1,2,...,m-e\}} U(h)'_k$.

L^e and M^e can be calculated by the following equations.

$$L_e = \sum_{j\in U_L^e} p_j \tag{27}$$

$$M_e = \max(\max_{j\in U_M^e} p_j, \text{Ceil}(\frac{1}{m}(S_e + \sum_{j\in U_M^e} size_j p_j))) \tag{28}$$

where S_e represents the lower bound for the total machine-time (the machine-time for task j is set as $p_j \cdot size_j$) of tasks in U_S^e that cannot be processed simultaneously with tasks in U_L^e. S_e is obtained through the following procedures:

Step 1: Set $S_e = 0$ and current number of machines $k = 1$;

Step 2: If $S_e + k \cdot \sum_{j\in U(h)'_k} p_j > k \cdot \sum_{j\in U(h)'_{m-k}} p_j$, update $S_e = S_e + k \cdot \sum_{j\in U(h)'_k} p_j - k \cdot \sum_{j\in U(h)'_{m-k}} p_j$; otherwise, update $S_e = 0$;

Step 3: Update $k = k + 1$. If $k > m - e$, go to step 4; otherwise, go to step 2;

Step 4: Output S_e.

We now prove LB_4^e is a valid lower bound for node h. Let P1 denote the problem which involves m machines with the same ready time g_1^h and tasks in $U_L^e \bigcup U_M^e \bigcup U_S^e$. Clearly, the lower bounds to P1 are also lower bounds for node h. Since no two tasks in U_L can be processed simultaneously, it should take a length of L_e time to finish all tasks from U_L^e. Further, due to the limitation of the total number of machines available, tasks belonging to U_M^e cannot be processed when any of the tasks in U_L^e is be processed. Therefore, the lower bound for the total machine-time of tasks belonging to $U_S^e \bigcup U_M^e$ that cannot be processed simultaneously with tasks in U_L^e can be calculated by $S_e + \sum_{j\in U_M^e} p_j$, and it is obvious that Ceil$(\frac{1}{m}(S_e + \sum_{j\in U_M^e} p_j))$ is a lower bound for the completion time of these tasks. Considering that $\max_{j\in U_M^e} p_j$ is also a lower bound for the completion time for these tasks, we can easily obtain that LB_4^e is a lower bound for P1 and therefore a lower bound for node h.

Observation 5.1. *LB_4 is valid only when $m > 3$.*
This is because U_L^e and U_S^e for any e are empty when $m \leq 3$, which easily follows that LB_4 is dominated by max(LB_1, LB_2).

Lower bound 5:

In Lower bound 5, we obtain the lower bound for the corresponding the $P|r_k|C_{\max}$ problem at node h. The $P|r_k|C_{\max}$ problem is solved by the LPT method, which was proved to have a performance guarantee of $1.5 - 2/m$ (Lee, 1991). Several properties are introduced below to specify conditions where the makespan provided by the LPT method is optimal for the $P|r_k|C_{\max}$ problem. In these properties, tasks and machines are indexed such that $p_1 \geq p_2 \geq ... \geq p_n$ and $r_1 \leq r_2 \leq ... \leq r_m$.

Theorem 5.1. *(Lee, 1991) Schedule the tasks by LPT. Let r be the index of the last job assigned to the machine on which the makespan M^* occurs. If $p_r > \frac{1}{2}M^*$ then the schedule obtained by LPT will yield the optimal makespan.*

Theorem 5.2. *(Webster, 1996) Let δ be an LPT schedule for the $P|r_k|C_{\max}$ problem with job start times, $s_1, ..., s_n$, and let t be the index of the last job to finish processing. If p_j/p_t is integer for all j satisfying $s_j < s_t$, then δ is optimal.*

Lemma 5.1. *(Webster, 1996) If $n < m$ and $r_n - r_1 < p_i$ for $i = 1, ..., n-1$, then the LPT rule is optimal.*

Assume the makespan yield by the LPT rule for the $P|r_k|C_{\max}$ problem is LM, then the fifth lower bound for node h is obtained by:

$$LB_5 = \begin{cases} LM & \text{if } LM \text{ is judged to be optimal by the above properties} \\ \text{Ceil}(LM/(1.5 - 2/m)) & \text{otherwise} \end{cases} \tag{29}$$

The time complexity of bounds LB_1, LB_2, LB_3, LB_4, and LB_5 is $O(n)$, $O(nm^2)$, $O(mn)$, $O(nm^2)$ and $O(n \log_2 n)$, respectively. For a newly generated node, the five lower bounds are calculated one by one, from LB_1 to LB_5, and the calculation stops if any of them is computed to be larger than or equal to the current upper bound, which effects a deletion decision of the node. If the five lower bounds are all smaller than current upper bound, then the largest one is recorded as the final lower bound for the node.

5.4. An illustrative example

In the section, we use an example to illustrate how the B & B method works. The example involves 5 machines and 10 tasks. The number of machines ($size_j$) and the processing time (p_j) required by each task are given in Table 2.

Table 2: Example data

task	1	2	3	4	5	6	7	8	9	10
$size_j$	1	2	1	4	5	3	3	2	4	2
p_j	23	71	17	20	80	41	75	80	6	52

5.4.1. Preprocessing and initial upper bound calculation

The solution procedure of the B & B method begins with preprocessing. In the example, $\hat{s} = 1$, therefore, we can remove tasks with $size_j > 4$ from N (in this example, task 5). After this step the 10-task problem reduces into a 9-task problem.

We then calculate the initial upper bound of the problem. The upper bounds given by the approximation algorithms proposed by this paper and Li (1999) are 216 and 288, respectively. Thereby, the initial upper bound is set to be 216.

5.4.2. Fathoming test

In this section, we illustrate how the fathoming test works by using the following 4 nodes, which are obtained after certain numbers of iterations in the search tree:

h_1: $D(h_1) = \{7, 10\}$, $U(h_1) = \{1, 2, 3, 6, 4, 8, 9\}$, $G(h_1) = \{52, 52, 75, 75, 75\}$;
h_2: $D(h_2) = \{1, 3, 4, 7, 9, 10\}$, $U(h_2) = \{2, 6, 8\}$, $G(h_2) = \{78, 78, 115, 115, 115\}$;
h_3: $D(h_3) = \{4, 10\}$, $U(h_3) = \{1, 2, 3, 6, 7, 8, 9\}$, $G(h_3) = \{20, 20, 20, 72, 72\}$;
h_4: $D(h_4) = \{4, 10\}$, $U(h_4) = \{1, 2, 3, 6, 7, 8, 9\}$, $G(h_4) = \{52, 72, 72, 72, 72\}$.

First, we illustrate how Property 5.1 works by using node h_1. Suppose a new node (denoted by h_1') is obtained by moving task 6 from $U(h_1)$ into $D(h_1)$, thus, by applying the LS rule, we have $D(h_1') = \{6, 7, 10\}$, $U(h_1') = \{1, 2, 3, 4, 8, 9\}$, and $G(h_1') = \{75, 75, 116, 116, 116\}$. It is easy to infer that at this node, the two machines with completion time 52 in $G(h_1)$ are kept idle for 23 unit times before they are scheduled to process task 6. Therefore, we have $BIT_6^1 = BIT_6^2 = 23$. Further, since task 1 in $U(h_1')$ has $size_1 = 1$ and $p_1 = 23 \leq BIT_6^1$, we can conclude that h_1' is dominated and can be fathomed.

We then use node h_2 to illustrate Property 5.2. Suppose a new node (denoted by h_2') is obtained by adding task 6 from $U(h_2)$ into $D(h_2)$. For h_2', we have $D(h_2') = \{1, 3, 4, 6, 7, 9, 10\}$, $U(h_2') = \{2, 8\}$, and according to the LS rule, $G(h_2) = \{115, 115, 156, 156, 156\}$. It can be easily obtained that two machines are kept idle for 37 unit times before they are reassigned to task 6 ($AIL_6 = 37$). Given that $size_7 = size_6$ and $p_6 < p_7 \leq p_6 + AIL_6$, we can conclude that h_2' is dominated and can be fathomed.

Finally, Property 5.3 is illustrated by using h_3 and h_4. It is easy to see that $U(h_3) = U(h_4)$, and $S_u = 1$. Besides, we have $g_k^{h_3} \leq g_k^{h_4}$ for each $1 \leq k \leq 5$. Therefore, h_4 is dominated by h_3 and should be fathomed.

5.4.3. Lower bound calculation

This section aims to show how the lower bounds are obtained for a node. To do this, we use the following node:

h_5: $D(h_5) = \{2\}$, $U(h_5) = \{1, 3, 4, 6, 7, 8, 9, 10\}$, $G(h_5) = \{0, 0, 0, 71, 71\}$.

Furthermore, suppose the current upper bound is 216. The lower bounds for h_5 are calculated as follows.

Lower bound 1:

First, we have $UT = \sum_{j \in U(h_5)} size_j p_j = 756$ and $DM = \sum_{k \in M} (g_5^{h_5} - g_k^{h_5}) = 213$. Then, since $UT > DM$, we obtain $LB_1 = g_5^{h_5} + \text{Ceil}((UT - DM)/5) = 180$.

Lower bound 2:

To begin with, μ =Floor(5/2)= 2. We then calculate CT_1 and CT_2. First, according to equation (19), it can be easily obtained that $CT_1^1 = 23$ and $CT_2^1 = 80$.

Besides, in the calculation of CT_1^2, a set $V(h_5)_1$ of $l = 5$ new tasks are first generated. These tasks all require 1 machine to process, and the processing times of these tasks are 0, 0, 0, 71 and 71. Hence, we can calculate $CT_1^2 = 0 + 0 + 0 = 0$. Similarly, we can obtain $CT_2^2 = 0 + 71 + 52 = 123$. Therefore, it follows that $CT_1 = 23$ and $CT_2 = 123$.

We then calculate CT_5, where tasks belonging to the set $U(h_5)_3 \bigcup U(h_5)_4$ are involved. Based on equation (20), we have $CT_5 = 142$. Therefore, $LB_2 = \max\{CT_1, CT_2, CT_5\} = 142$.

Lower bound 3:

To obtain LB_3, we first calculate LB_3^1, which, according to equation (23), is equal to 146(0 + 75 + 71). Then, after the construction of $U(h_5)'$, LB_3^2 can be calculated by the following procedure:

Step 1: Set $om = 0$ and $w_k = 0$, $\forall k \in M$;

Step 2: Since $om < m$, identify the first task J ($size_J = 2$ and $p_J = 80$);

Step 3: Since $om + size_J \leq m$, update $w_k = p_J = 80$, $k = 1, 2$, and set $om = om + size_J = 2$;

Step 4: Since $om < m$, identify the second task J ($size_J = 3$ and $p_J = 75$);

Step 5: Since $om + size_J \leq m$, update $w_k = p_J = 75$, $k = 3, 4, 5$, and set $om = om + size_J = 5$;

Step 6: Since $om = m$, calculate $LB_3^2 = 0 + 146 = 146$.

Lower bound 4:

This part calculates LB_4 for node h_5. To do this, we first construct sets $U(h)'_k$ where $k \in \{2, 3, 4, 5\}$. Then, for $e \in \{3, 4, 5\}$ (note that $\mu = 2$), sets U_L^e, U_M^e and U_S^e are generated. We are now ready to calculate LB_4^e, where $e \in \{3, 4, 5\}$. For example, $LB_4^4 = 180$, which is calculated by using $L_4 = 26$, $M_4 = 154$ and $S_4 = 14$. Note that S_4 can be obtained by the following procedure:

Step 1: Set $S_4 = 0$ and current number of machines $k = 1$;

Step 2: Since $0 + 1 \cdot \sum_{j \in U(h)'_1} p_j = 40 > 1 \cdot \sum_{j \in U(h)'_4} = 26$, update $S_e = 0 + 40 - 26 = 14$;

Step 3: Update $k = 2$;

Step 4: Since $k = 2 > m - e = 1$, output $S_e = 14$.

Similarly, we have $LB_4^3 = 180$ and $LB_4^5 = 180$, which easily follows that $LB_4 = 180$.

Lower bound 5:

In this part, the corresponding the $P|r_k|C_{\max}$ is first solved by the LPT method, leading to the makespan of 183. Then, we find that obtained makespan cannot be judged to be optimal for the $P|r_k|C_{\max}$ problem according to Theorem 5.1, Theorem 5.2 and Lemma 5.1. Therefore, we have $LB_5 = \text{Ceil}(183/(1.5 - 2/5)) = 131$.

We have now finished calculating all the five lower bounds for node h_5. Since they are all less than the current upper bound, we choose the largest one of them (LB_1) to be the final lower bound for h_5.

5.4.4. Upper bound update

Suppose the current upper bound is 216 and we have the following node:

h_6: $D(h_6) = \{1, 2, 3, 6, 7, 8, 9, 10\}$, $U(h_6) = \{4\}$, $G(h_6) = \{173, 179, 179, 179, 179\}$.

A new node (denoted by h'_6) can be obtained by adding task 4 into $D(h_6)$. It is obvious that h'_6 corresponds to a new solution of the problem whose makespan is 199. As the new makespan is less than the current upper bound, we update the upper bound to be 199.

6. A tabu search algorithm

Since the solution time of the branch and bound method grows exponentially with the increase of the size of problems, it is expected to be inefficient for large-sized instances. Therefore, we provide a tabu search algorithm (TS) to solve the $P|size_j|C_{\max}$ problem with large sizes. TS is a local search-based meta-heuristic designed to find near-optimal solutions for combinatorial optimization problems. The method was originally proposed by Glover (1986) and has been widely applied in solving practical optimization problems (e.g., França et al. (1996); Solimanpur and Elmi (2013); Mensendiek et al. (2015)).

6.1. Solution representation and initiation

The $P|size_j|C_{\max}$ problem can be decomposed into three subproblems:

- a *sequence* problem, which determines the priorities among tasks;
- an *assignment* problem, which assigns tasks to a given number of machines;
- a *scheduling* problem, which determines the starting time of the tasks based on machine assignment and task sequence.

The proposed TS is designed to solve the first subproblem of the $P|size_j|C_{\max}$ problem to obtain a sequence σ of all tasks, while the subsequent assignment and scheduling problems are then handled by the LS rule to yield a schedule denoted by $\mathcal{L}(\sigma)$ corresponding to σ. In the following part of this section, "sequence" and "solution" will be used interchangeably.

In TS, an initial solution is required to provide the starting point for the local search procedure. Our algorithm generates an initial solution (σ_0) by randomly sequencing all tasks of the $P|size_j|C_{\max}$ problem. Like in the B & B algorithm, to reduce the search space, only tasks with $size_j \leq m - \hat{s}$, where $\hat{s} = \min_{j \in N} size_j$, are considered in the TS. Therefore, the makespan yielded by the TS should be added by the summation of processing times of tasks with $size_j > m - \hat{s}$ to get the complete makespan.

6.2. Neighborhood structure

TS is by nature a local-search-based optimization algorithm and in each iteration, it searches in the neighborhood of the current solution to detect a new solution which will then replace the current one.

Before explaining how the neighborhood is constructed, we introduce the concept of *disjunctive graph model* which is applied to represent and better analyze solutions to the $P|size_j|C_{\max}$ problem. The *disjunctive graph model* was proposed by Roy and Sussmann (1964). In the *disjunctive graph*, all tasks are represented as nodes. The processing sequence of a machine is represented as a chain consisting of a set of conjunctive arcs which indicates a path from a dummy source node 0 to a dummy sink node T. The lengths of these arcs are defined as the processing time of the task they lead to. In addition, for the node representing task j, the numbers of conjunctive arcs that direct to and from it should both equal $size_j$. Apparently, the makespan for a certain schedule of the $P|size_j|C_{\max}$ problem in the *disjunctive graph* context is represented as the length of the *longest path* linking node 0 and node T.

Figure 2 gives an example of the disjunctive graph model for the $P|size_j|C_{\max}$ problem with 10 tasks and 3 machines. In this example, the solution to the problem is given by the task sequence $1-4-2-3-6$ for machine 1, $5-2-9-7$ for machine 2 and $4-10-8-7$ for machine 3.

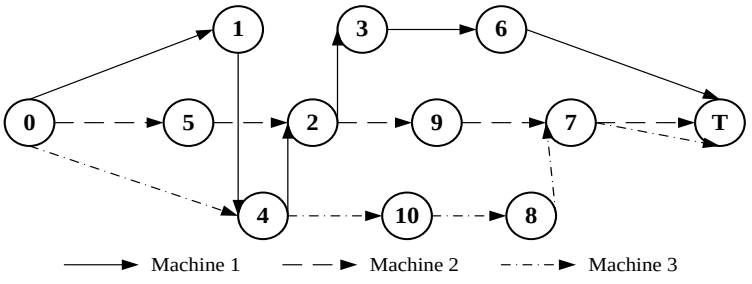

Figure 2: An illustration of the disjunctive graph representation.

We now introduce the neighborhood structure ($N(\sigma)$) for solution (σ) which is constructed in a retrospective way, as shown in Algorithm 1.

In the following pseudocode, for each task i, we use $\bar{p}(i)$ to denote the assigned processor with the largest ready time for the task. By ready time we mean the time when a machine finishes all the tasks assigned to it before task i in a sequence. Further, let $\bar{t}(i)$ denote the last task completed by machine $\bar{p}(i)$ before it starts to handle task i. If task i is the first task assigned to $\bar{p}(i)$, we denote $\bar{t}(i) = 0$.

Algorithm 1 The neighborhood construction procedure

Input: (1) Current Schedule: $\mathcal{L}(\sigma)$; (2) Maximum Searching Length MS;
Output: Neighborhood $N(\sigma)$ for $\mathcal{L}(\sigma)$;

```
N(σ) = ∅;
For each k ∈ M, initiate partial handling sequence of machine k (denoted by s̄(k)) to be ∅;
Initiate i to be the last finished task in L(σ);
while t̄(i) ≠ 0 do
    Identify p̄(i) and t̄(i);
    Add t̄(i) as the last task into sequence s̄(k);
    i = t̄(i);
end while
for k = 1 to |M| do
    Get σ' by swapping the sequences of any two tasks if the sequential difference between them in s̄(k) is no more than MS;
    Add σ' into N(σ);
end for
```

For example, as shown in Figure 2, assume $\sigma = 1-5-4-3-2-10-6-9-8-7$ is the current solution for a problem with 10 tasks and 3 machines. Further, assume $\bar{s}(2)$ generated by the above procedure is $5-2-9$. We then obtain a new sequence $\sigma' = 1-2-4-3-5-10-6-9-8-7$ by swapping the positions of task 5 and 2 in σ and add it into $N(\sigma)$.

6.3. Stop criteria

In order to strike a balance between the solution quality and speed, two stop criteria are adopted in the algorithm. The algorithm stops (1) if no better solution has been found in a continuous $\eta_{\max}$ iterations or (2) if the algorithm has completed $\lambda_{\max}$ iterations.

6.4. Search procedure

Before proposing the search procedure of the TS, it is necessary to introduce several notations. First, as Cordeau et al. (1997) and Sammarra et al. (2007) did in their studies, we describe the schedule $\mathcal{L}(\sigma)$ by a set of attributes

$$B(\sigma) = \{(i, j, k) | \text{ machine } k \text{ starts processing task } j \text{ immediately after it finishes task } j \text{ in } \mathcal{L}(\sigma)\}.$$

Then, let PM be the punishment multiplier for generating repeating attributes when a new solution is created to replace the current one and TL be the tabu length. The iteration within which an attribute (i, j, k) is forbidden is denoted by α_{ij}^k. The aspiration level of an attribute is denoted by β_{ij}^k. Further, the frequency of an attribute being newly generated in solutions is denoted by γ_{ij}^k. Obviously, any transition between σ and σ' in $N(\sigma)$ will correspondingly get $B(\sigma)$ replaced by $B(\sigma')$ where some attributes are removed and some new ones are added. Moreover, we use λ to record the current iteration and $\lambda_{\max}$ to denote the total number of iterations.

We are now ready to introduce the search procedure of the TS, which is given in Algorithm 2. Detailed explanations about some steps in the algorithm are given below:

- Some transitions from σ to $\sigma' \in N(\sigma)$ may generate new attributes (i, j, k) that are still in tabu, these transitions should be forbidden. However, if σ' yields a shorter makespan than σ,

Algorithm 2 The search procedure of the proposed TS.

Input: (1) Problem data: N, M, $size_j$,p_j and (2) Controlling parameters: PM, TL, MS, $\eta_{\max}$ and $\lambda_{\max}$;
Output: Best $C^*_{\max}$ and best sequence σ^* found so far;
$\forall\ (i,j,k)$, $\alpha^k_{ij}=0, \beta^k_{ij}=\infty, \gamma^k_{ij}=0$;
Initiate the number of iterations within which no better solution is found $\eta=0$;
Initiate task sequence σ_0. (Section 6.1);
σ=σ_0;
$\sigma^*=\sigma$, $C^*_{\max}$=$C_{\max}(\sigma)$;
$\forall\ (i,j,k)\in\mathcal{L}(\sigma), \beta^k_{ij}=C^*_{\max}$;
for $\lambda=\{1,...,\lambda_{\max}\}$ **do**
  $\eta=\eta+1$;
  Construct neighbour $N(\sigma)$ (Section 6.2);
  $M(\sigma)=\varnothing$;
  $\forall\sigma'\in N(\sigma)$, $\forall(i,j,k)\in B(\sigma')\setminus B(\sigma)$, if $\alpha^k_{ij}<\lambda$ or $C_{\max}(\sigma')<\beta^k_{ij}$, $M(\sigma)=M(\sigma)\bigcup\sigma'$;
  $\forall\sigma'\in M(\sigma)$, if $C_{\max}(\sigma')<C_{\max}(\sigma)$, $v(\sigma')=C_{\max}(\sigma')$; else, $v(\sigma')=C_{\max}(\sigma')+PM\cdot\sum_{(i,j,k)\in B(\sigma')\setminus B(\sigma)}\gamma^k_{ij}$;
  $\hat{\sigma}$=arg $\min_{\sigma'\in M(\sigma)} v(\sigma')$;
  $\forall(i,j,k)\in B(\hat{\sigma})\setminus B(\sigma)$, $\gamma^k_{ij}=\gamma^k_{ij}+1$;
  $\forall(i,j,k)\in B(\sigma)\setminus B(\hat{\sigma})$, $\alpha^k_{ij}=\lambda+TL$;
  $\forall(i,j,k)\in B(\hat{\sigma})$, if $\beta^k_{ij}>C_{\max}(\hat{\sigma})$, $\beta^k_{ij}=C_{\max}(\hat{\sigma})$;
  if $C^*_{\max}>C_{\max}(\hat{\sigma})$, $C^*_{\max}=C_{\max}(\hat{\sigma})$, $\sigma^*=\hat{\sigma}$ and $\eta=0$;
  $\sigma=\hat{\sigma}$.
  if $\eta\geq\eta_{\max}$, break the for loop;
end for

σ' should be considered as a valid neighboring solution for σ, no matter whether there are attributes of σ' that are held in tabu or not. This is how $M(\sigma)$ is derived from $N(\sigma)$;

- In the algorithm, each σ' is evaluated in terms of both the derived makespan and the level of originality. Therefore, a punishment is added to $v(\sigma')$ if some attributes that are newly added into $B(\sigma')$ have been generated before. However, σ' with $C_{\max}(\sigma')$ smaller than the current $C_{\max}(\sigma)$ is exempted from such punishments;
- σ' with the best $v(\sigma')$ (denoted by $\hat{\sigma}$) is selected to replace σ to enter the next iteration, while attributes in $B(\sigma')$ which are newly generated are declared tabu for the next TL iterations.

7. Numerical experiments

In order to examine the performances of the proposed models and solution methods, a large number of computational experiments are conducted. According to their sizes, the experiments are divided into five parts (Part A, B, C, D, and E), with 180, 300,150, 40 and 30 instances respectively. In all parts, we solve the instances by the proposed B & B method and the TS, and in the first part, the 180 instances are solved by the B & B method proposed by Blkadek et al. (2015) as well. Note that for notational convenience, in the following part of this paper, we denote the B & B method proposed by Blkadek et al. (2015) as BLK and the "B & B method" is therefore used exclusively to refer to the B & B method proposed by this paper. Besides, the instances in Parts A and B are also solved by CPLEX using the two models developed in Section 4. The algorithms are coded in $C++$ language, and CPLEX 12.6 is used to solve instances in the first two parts. The experiments are conducted on an Intel Core *i*7 3.60 GHz PC with 16 GB RAM.

7.1. Instance generation and algorithm settings

The 700 instances for experiments are generated in a random way. For an instance with n tasks and m machines, the n tasks are generated independently, each assigned with the processing time

generated from a uniform distribution of $U(3, 80)$ and the required number of machines generated randomly within the range of $[1, m]$.

We now get the solution methods prepared. To begin with, the time limit of the B & B method for all instances is set to be 3600 s, and the same time limit is also set for CPLEX for solving instances in Part A and B. Then, the parameters used in the TS are set as follows. First, the punishment multiplier PM, and the tabu length TL are set as $0.1\sqrt[3]{nm^2}$, 13, respectively, for all instances. Second, to strike a balance between solution quality and speed, the maximum searching length MS is set adaptively, according to the sizes of instances. For instances with $n \leq 100$ we set $MS = |N|$ so as to enable the algorithm to search in neighbourhoods as large as possible, for instances with $100 < n \leq 150$ MS is set as $40 - m$, and for instances with $n > 200$, MS is fixed to be 1. Third, the stop criteria (including parameters $\eta_{\max}$ and $\lambda_{\max}$) are also set adaptively. For instances with $n \leq 150$, $\eta_{\max}$ and $\lambda_{\max}$ are set to be 1000 and 5000, respectively, and for instances with $n \geq 250$ we set $\eta_{\max} = 500$ and $\lambda_{\max} = 2500$. Further, the TS is run 30 times to solve each instance.

7.2. Lower bound calculation

To better assess the results obtained by the algorithms, we provide lower bounds (LB) for instances that our proposed B & B method, the BLK or the CPLEX (using the proposed models) cannot solve optimally within 3600 s. For each of these instance, the lower bound is computed in two steps: first, a lower bound for the makespan of tasks with $size_j \leq m - \hat{s}$, where $\hat{s} = \min_{j \in N} size_j$, is calculated according to the description in section 5.3.2; then the summation of processing time of tasks with $size_j > m - \hat{s}$ is added to the lower bound to obtain the final lower bound. Hence, gaps for solutions (f) obtained by the two B & B methods, CPLEX and the TS for each subset can be calculated as:

$$Gap = (f - f_{LB})/f_{LB} \times 100. \tag{30}$$

where f_{LB}=min$\{f_{BB}, f_{BLK}, f_{CPLEX}\}$, if the proposed B & B method, the BLK or the CPLEX is able to deliver the optimal solution (denoted by f_{BB}, f_{BLK}, and f_{CPLEX}, respectively), and f_{LB}=LB, if not.

7.3. Results of instances with small sizes

We first test the performances of the proposed models (by using CPLEX) and algorithms for solving instances with small sizes. The instances in Part A, Part B, and Part C are used in this section. In Part A, instances involving 3 to 5 machines and 5 to 10 tasks are further divided into 18 subsets (marked as A01-A18).In Part B, instances involving 3 to 5 machines and 11 to 20 tasks are further divided into 30 subsets (marked as B01-B30). Besides, in Part C, instances involving 6 to 10 machines and 20 to 40 tasks are further divided into 15 subsets (marked as C01-C15). Each subset contains 10 instances with the same numbers of machines and tasks.

7.3.1. Results of instances in Part A

For all instances in this part, the proposed B & B method and the BLK method manage to obtain the optimal solutions, and the results of every instance in this part obtained by the CPLEX and the TS are evaluated by their gaps (in percentage) against f_{LB} obtained in Section 7.2. Table 3 compares performances of these methods for solving instances in Part A. In this table, performances of these methods are reported corresponding to the partition into 18 subsets. Details of the table are given below.

For each subset, the average of Gaps of solutions of the 10 instances in a subset obtained by the two models using CPLEX is shown in columns "AG" of "M1" and "M2". Columns "US" of "M1" and "M2" show the numbers of instances in a subset that cannot be solved optimally within

3600 s by these methods. In addition, as the CPLEX using model M1 fails to deliver feasible solutions for some of the instances, the column "NS" reports the number of such instances in a subset. To evaluate the performances of TS, we first calculate the *Gap*s of the average solution, the best solution and the worst solution of the 30 runs for each instance which are denoted by Gap_a, Gap_b, and Gap_w, respectively. Moreover, the standard deviation of the *Gap*s of the 30 runs for each instance is also computed (denoted by Gap_{sd}). Then, the averages of Gap_as, Gap_bs, Gap_ws, and Gap_{sd}s of 10 instances in a subset is reported in columns "AG", "BG", "WG" and "SD" in "TS", accordingly. Besides, in columns "Time" of "B & B", "M1" and "M2", we report the average solution time in seconds needed by them to solve 10 instances in different subsets, and the column "Time" of "TS" shows the average time of all the 300 (10×30) runs in a subset.

Table 3: Results of small-sized instances in Part A.

Subset	m	n	B & B	BLK	M1				M2			TS				
			Time(s)	Time(s)	AG(%)	Time(s)	US	NS	AG(%)	Time(s)	US	AG(%)	BG(%)	WG(%)	SD(%)	Time(s)
A01	3	5	0.00	0.00	0.00	0.42	0	0	0.00	0.68	0	0.00	0.00	0.00	0.00	0.00
A02	3	6	0.00	0.00	0.00	5.70	0	0	0.00	2.03	0	0.75	0.00	3.85	1.32	0.00
A03	3	7	0.00	0.01	0.00	992.72	1	0	0.00	7.50	0	0.16	0.00	4.47	0.83	0.01
A04	3	8	0.00	0.04	0.00	2800.11	7	0	0.00	26.20	0	0.57	0.00	3.89	1.01	0.00
A05	3	9	0.00	0.31	0.08	3600.00	10	0	0.00	60.70	0	0.45	0.00	4.27	0.95	0.01
A06	3	10	0.00	3.60	0.73	3600.00	10	0	0.00	1819.65	5	0.17	0.00	0.57	0.24	0.01
A07	4	5	0.00	0.00	0.00	0.51	0	0	0.00	0.76	0	0.67	0.00	3.89	1.45	0.00
A08	4	6	0.00	0.00	0.00	90.50	0	0	0.00	3.90	0	0.21	0.00	0.71	0.33	0.00
A09	4	7	0.00	0.01	0.00	2195.97	5	0	0.00	7.58	0	0.42	0.00	5.25	1.16	0.01
A10	4	8	0.00	0.05	0.28	3600.00	10	0	0.00	14.57	0	0.50	0.00	3.76	1.00	0.00
A11	4	9	0.00	0.51	0.00	3600.00	10	0	0.00	456.54	0	0.07	0.00	0.36	0.11	0.01
A12	4	10	0.01	3.64	4.82	3600.00	10	0	0.00	1187.01	3	0.37	0.00	1.99	0.65	0.02
A13	5	5	0.00	0.00	0.00	0.65	0	0	0.00	0.60	0	1.14	0.00	3.04	1.44	0.00
A14	5	6	0.00	0.00	0.00	540.01	1	0	0.00	1.77	0	0.38	0.00	3.93	1.03	0.00
A15	5	7	0.00	0.01	0.00	3000.19	8	0	0.00	10.20	0	0.07	0.00	0.76	0.20	0.00
A16	5	8	0.00	0.06	0.10	3600.00	10	0	0.00	7.70	0	0.53	0.00	2.91	0.89	0.01
A17	5	9	0.00	0.36	1.21	3600.00	10	0	0.00	16.66	0	0.26	0.00	3.79	0.74	0.01
A18	5	10	0.01	3.91	4.80	3600.00	10	2	0.00	910.27	2	0.25	0.00	3.31	0.66	0.02
Average			0.00	0.70	0.67	2134.82	5.67	0.11	0.00	251.91	0.56	0.39	0.00	2.82	0.78	0.01

Note 1. "BLK" refers to the B & B method proposed by Blkadek et al. (2015).

Note 2. Values in "M1" are calculated based on instances for which the CPLEX using model M1 is able to deliver integer solutions.

Note 3. "AG" reports average Gaps, "BG" reports average best Gaps and "SD" reports average standard deviations.

Note 4. "US" reports the number of unsolved instances and "NS" reports the number of instances for which no feasible solutions are reported.

We first compare the efficiencies of the proposed models. As is shown in Table 3, for most subsets, the time-discretized model M2 demonstrates fewer unsolved instances and shorter solution times. In addition, it is able to find feasible solutions for all of the 180 instances. Therefore, it is clear that M2 outperforms the VRP-based Model M1 for most cases in terms of both solution quality and speed. However, when m and n are relatively small, M1 can better solve subsets A01 and A07.

We then look at the performances of the proposed B & B method. To begin with, three methods (the proposed B & B method, the BLK method and the CPLEX using M2) manage to produce optimal solutions for all of the instances. Besides, all values listed in columns "AG" of M1 and the TS are no less than and a considerable proportion of the values are larger than the "AG" of the B & B algorithm; this implies that for most instances the solutions obtained by the CPLEX using M1 and the average solutions delivered by the TS are worse than the solutions provided by the B & B method. What is more, as for the solution speed, the B & B method represents smaller average solution times than all the other methods for all subsets. Therefore, the proposed B & B method

outperforms other methods when solving instances in Part A.

The TS can also well solve instances in this part. To begin with, the near-zero values in column "AG" and "SD" for all subsets imply that the TS is able to produce a solution with very high qualities in each run. In addition, the all zero values in column "BG" indicate that the algorithm can obtain the optimal solution for all the 180 instances within 30 runs. Besides, the all-small values in column "WG" demonstrate the robustness of the algorithm. Moreover, the TS can solve these instances very quickly—no average time for solving instances in a subset goes larger than 0.02s.

7.3.2. Results of instances in Part B

The section tests instances from Part B which involves 3 to 5 machines and 11 to 20 tasks. We solve these instances by the B & B method, the CPLEX (using M1 and M2) and the TS. Note that the BLK method is no longer applicable for solving instances in this part, since the "out of memory" issue is reported even when using the BLK to solve the smallest instances in Part B. Table 4 demonstrates the results of these methods for solving instances in Part A, which are reported corresponding to the partition into 30 subsets.

We first compare the efficiencies of the proposed models. As shown in the table, M1 reaches the time limit (3600 seconds) for solving all instances and fails to deliver feasible solutions for a considerable part of the instances. In comparison, M2 reports better performance, as it shows less average solution times for most instances and manages to find feasible solutions for all instances.

The B & B method can produce optimal solutions to 299 of all 300 instances, and it can solve instances with up to 15 tasks and 5 machines in very short times (most instances can be solved optimally in less than 1 second). Further, all values listed in columns "AG" of M1, M2, and the TS are no less than and a considerable proportion of the values are larger than the "AG" of the B & B algorithm; this implies that for most instances the solutions obtained by the CPLEX and the average solutions delivered by the TS are worse than the solutions provided by the B & B method. Hence, when it comes to the solution quality, the B & B method outperforms all the other methods. As for the solution speed, the B & B method represents smaller average solution times than M1 and M2 for all subsets. In addition, although the B & B method is slower than the TS for some subsets, the differences are all relatively narrow for most subsets.

The TS also demonstrates its efficiency for solving instances in Part B. First, the values in column "AG" and "SD" for all subsets are all small, which implies that the algorithm is able to produce a solution with very high qualities in each run. Second, the zero values in column "BG" for the majority of subsets indicate that the algorithm manages to obtain the optimal solution for most instances within 30 runs. Third, the all-small values in column "WG" indicate that the method is of great robustness. Moreover, the TS records the best solution speed for solving instances in this part (the average time is only 0.08 seconds).

7.3.3. Results of instances in Part C

We further examine the performances of the B & B method and TS for solving the small-sized instances in Part C. The 150 instances with 6 to 10 machines and 20 to 40 tasks are tested in this section. Like in the previous section, the 150 instances are partitioned into 15 subsets according to the numbers of machines and tasks, and we report by averaging the computational results of instances in a subset. The results for instances from Part C are given in Table 5 where notations bear the same meanings as in Table 3 and 4 .

As can be seen in Table 5, the average computing time and the number of unsolved instances of the B & B method show general increasing trends with growing problem sizes. In addition, there are more than half of the instances in this part that cannot be solved optimally within 3600 s by the B & B algorithm. Nevertheless, the average solution gaps of the B & B method are all small. This implies that the algorithm can obtain optimal or near-optimal solutions for most instances in this part within a reasonable time.

Table 4: Results of small-sized instances in Part B.

Subset	m	n	B & B			M1				M2			TS				
			AG(%)	Time(s)	US	AG(%)	Time(s)	US	NS	AG(%)	Time(s)	US	AG(%)	BG(%)	WG(%)	SD(%)	Time(s)
B01	3	11	0.00	0.00	0	1.36	3600.00	10	0	0.00	2207.51	6	0.36	0.00	1.79	0.41	0.01
B02	3	12	0.00	0.00	0	2.03	3600.00	10	0	0.00	2530.34	7	0.20	0.00	2.17	0.47	0.01
B03	3	13	0.00	0.09	0	4.27	3600.00	10	0	0.02	1473.93	3	0.13	0.00	0.70	0.17	0.03
B04	3	14	0.00	0.01	0	10.10	3600.00	10	2	0.00	1392.42	2	0.03	0.00	0.84	0.15	0.03
B05	3	15	0.00	0.04	0	14.88	3600.00	10	2	0.00	1525.37	3	0.23	0.00	1.18	0.34	0.03
B06	3	16	0.00	0.21	0	12.68	3600.00	10	4	0.02	2012.63	5	0.09	0.00	1.53	0.33	0.05
B07	3	17	0.00	0.19	0	43.00	3600.00	10	7	0.05	2186.60	4	0.08	0.00	0.61	0.12	0.06
B08	3	18	0.00	2.39	0	NONE	3600.00	10	10	0.23	3113.54	7	0.09	0.00	0.72	0.16	0.07
B09	3	19	0.00	1.13	0	NONE	3600.00	10	10	0.34	3249.26	7	0.03	0.00	0.18	0.05	0.06
B10	3	20	0.00	34.19	0	31.59	3600.00	10	9	0.11	2586.71	6	0.01	0.00	0.14	0.03	0.17
B11	4	11	0.00	0.00	0	4.53	3600.00	10	0	0.00	1853.60	4	0.20	0.00	3.49	0.70	0.02
B12	4	12	0.00	0.00	0	8.46	3600.00	10	0	0.00	2156.33	5	0.12	0.00	2.62	0.53	0.03
B13	4	13	0.00	0.04	0	15.11	3600.00	10	0	0.04	3098.27	7	0.16	0.00	0.93	0.22	0.04
B14	4	14	0.00	0.02	0	20.38	3600.00	10	3	0.17	3600.00	10	0.20	0.00	1.82	0.40	0.05
B15	4	15	0.00	0.32	0	18.15	3600.00	10	5	0.24	3266.33	9	0.33	0.00	2.02	0.43	0.07
B16	4	16	0.00	0.19	0	27.39	3600.00	10	7	0.49	3600.00	10	0.54	0.00	1.70	0.43	0.09
B17	4	17	0.00	2.18	0	57.44	3600.00	10	9	0.26	3599.75	9	0.09	0.00	0.79	0.18	0.09
B18	4	18	0.00	0.07	0	NONE	3600.00	10	10	0.39	3600.00	10	0.18	0.00	0.66	0.18	0.09
B19	4	19	0.00	5.53	0	NONE	3600.00	10	10	0.65	3485.92	8	0.52	0.05	1.55	0.32	0.13
B20	4	20	0.00	5.61	0	NONE	3600.00	10	10	0.29	3600.00	10	0.21	0.02	0.77	0.18	0.15
B21	5	11	0.00	0.00	0	4.78	3600.00	10	1	0.00	1576.13	4	0.41	0.00	2.61	0.69	0.02
B22	5	12	0.00	0.01	0	9.20	3600.00	10	0	0.00	1196.32	3	0.41	0.05	3.95	0.76	0.04
B23	5	13	0.00	0.01	0	15.33	3600.00	10	1	0.09	2050.11	5	0.33	0.00	1.25	0.32	0.05
B24	5	14	0.00	0.04	0	14.37	3600.00	10	4	0.03	1986.76	5	0.23	0.00	1.91	0.41	0.09
B25	5	15	0.00	0.02	0	32.61	3600.00	10	7	0.30	1833.31	4	0.17	0.03	1.94	0.37	0.08
B26	5	16	0.00	2.70	0	NONE	3600.00	10	10	0.00	1844.60	3	0.11	0.00	0.44	0.13	0.14
B27	5	17	0.00	5.17	0	27.10	3600.00	10	8	0.15	2422.55	6	0.28	0.00	2.10	0.42	0.14
B28	5	18	0.00	11.80	0	NONE	3600.00	10	10	0.35	3020.68	8	0.49	0.11	1.16	0.25	0.14
B29	5	19	0.00	229.67	0	NONE	3600.00	10	10	0.58	3418.34	9	0.36	0.08	0.89	0.19	0.15
B30	5	20	0.03	368.88	1	NONE	3600.00	10	10	0.19	2443.08	6	0.19	0.05	0.50	0.12	0.21
Average			0.00	22.35	0.03	17.85	3600.00	10.00	5.30	0.17	2531.01	6.17	0.23	0.01	1.43	0.32	0.08

Note 1. Values in "M1" are calculated based on instances for which the CPLEX using model M1 is able to deliver integer solutions.

Note 2. Values in "M1" are displayed as "None" if CPLEX using model M1 fails to deliver integer solutions for all instances in a subset.

Note 3. "AG" reports average Gaps, "BG" reports average best Gaps and "SD" reports average standard deviations.

Note 4. "US" reports the number of unsolved instances and "NS" reports the number of instances for which no feasible solutions are reported.

Table 5: Results of small-sized instances in Part C.

Subset	m	n	B & B			TS				
			AG(%)	Time(s)	US	AG(%)	BG(%)	WG(%)	SD(%)	Time(s)
C01	6	20	0.26	831.14	2	0.70	0.34	1.08	0.20	0.21
C02	6	25	0.58	1717.41	4	1.16	0.67	1.85	0.25	0.44
C03	6	30	1.08	2167.35	6	1.03	0.56	1.61	0.29	0.70
C04	6	35	1.05	2521.92	7	1.22	0.41	2.94	0.61	1.34
C05	6	40	1.04	2551.10	7	1.31	0.36	2.95	0.60	1.96
C06	8	20	0.84	1365.56	3	2.00	1.24	3.58	0.52	0.25
C07	8	25	1.18	2250.46	6	1.84	1.09	2.51	0.33	0.62
C08	8	30	1.52	2519.96	7	1.66	1.06	3.76	0.65	1.26
C09	8	35	1.37	2160.17	6	0.87	0.45	2.11	0.41	1.58
C10	8	40	1.45	2520.04	7	1.87	1.25	3.35	0.49	2.26
C11	10	20	0.24	1073.11	2	1.00	0.38	2.44	0.44	0.45
C12	10	25	1.65	2316.62	6	2.48	1.43	4.13	0.55	0.75
C13	10	30	1.95	2879.98	8	2.20	1.41	3.74	0.56	1.36
C14	10	35	2.77	3240.02	9	2.90	1.89	4.77	0.71	2.34
C15	10	40	3.33	3272.33	9	2.84	1.81	4.14	0.58	2.56
Average			1.35	2225.81	5.93	1.67	0.96	3.00	0.48	1.21

Note. "AG" reports average Gaps, "BG" reports average best Gaps and "SD" reports average standard deviations.

We now look at the results of the TS. To begin with, for 4 of the 15 subsets, the TS shows better values in "AG". Besides, the all-small values reported in "SD" attest the robustness of the TS. In addition, the values in "BG" reported by the TS are less than the values in "AG" reported by the B & B method for all subset except C01, C02, C06 and C11 where the instance sizes are relatively small. Further, no value in "WG" goes larger than 5%. These indicate that for the majority of the instances, the TS can generate high-quality solutions in each run and is able to find solutions better than those obtained by the B & B method within 30 runs. Moreover, the TS solves all instances in very short times. Therefore, the TS can well solve the instances in Part C and outperforms the B & B method on average.

7.4. Results of instances with large sizes

This section tests large-sized instances in Part D and Part E. We solve these instances by the B & B algorithm and the TS. The computational results are shown in Table 6 and Table 7, where the solutions yielded by the algorithms are evaluated by their gaps against the lower bounds calculated by equation (30). In these tables, we report for each instance the gap of the solution obtained by the B & B method (Gap), and for the 30 runs of the TS for each instance, we demonstrate the average gap (Gap_a), the smallest gap (Gap_b), the largest gap (Gap_w) and the standard deviation of the 30 gaps (Gap_{sd}). Further, the solution time of the B & B algorithm and the average solution time of the TS for every instance are also given in the two tables.

7.4.1. Results of instances in Part D

Table 6 covers the results of the B & B algorithm and the TS for solving 35 instances in Part D which involve 15 to 30 machines and 60 to 150 tasks.

As shown in Table 6, the B & B method reaches its time limit (3600 s) and fails to provide optimal solutions for all instances in this part. Nevertheless, the solutions obtained by the algorithm still demonstrate high qualities, as the average gap against the lower bounds is only 2.68% and no solution gap goes larger than 6%.

The TS shows better performance for solving the instances in Part D. First, the TS solves all instances in much shorter times. In addition, it manages to deliver optimal solutions for at least 3 of the 35 instances, and for most instances, the TS reports better solutions than the B & B method

Table 6: Results of large-sized instances in Part D.

Instance No.	m	n	B & B		TS				
			Gap(%)	Time(s)	$Gap_a(\%)$	$Gap_b(\%)$	$Gap_w(\%)$	$Gap_{sd}(\%)$	Time(s)
1	15	60	3.46	3600.00	4.12	3.39	4.95	0.43	14.76
2	15	70	5.62	3600.00	6.15	3.70	8.42	1.15	33.14
3	15	80	5.12	3600.00	5.14	2.65	8.35	1.47	61.17
4	15	90	2.39	3600.00	4.78	3.18	5.79	0.66	65.15
5	15	100	2.66	3600.00	0.64	0.00	2.09	0.59	117.06
6	15	110	0.48	3600.00	0.70	0.00	2.50	0.64	77.27
7	15	120	4.65	3600.00	5.59	4.07	8.10	1.08	123.09
8	15	130	1.91	3600.00	4.97	2.94	6.98	1.02	218.42
9	15	140	2.39	3600.00	4.73	2.78	6.26	0.83	179.67
10	15	150	1.39	3600.00	4.83	2.67	7.24	1.25	229.35
11	20	60	5.28	3600.00	5.79	3.10	8.80	1.31	35.54
12	20	70	2.88	3600.00	2.84	1.78	4.37	0.58	53.11
13	20	80	0.05	3600.00	2.14	0.32	5.03	1.04	124.23
14	20	90	3.65	3600.00	5.59	4.20	9.67	1.14	120.92
15	20	100	4.14	3600.00	4.95	3.29	6.63	0.81	146.39
16	20	110	2.25	3600.00	3.44	2.50	4.79	0.60	75.70
17	20	120	3.22	3600.00	6.11	4.04	7.59	0.87	171.83
18	20	130	1.65	3600.00	3.35	0.72	5.79	1.16	230.06
19	20	140	1.66	3600.00	5.89	3.55	10.14	1.21	239.40
20	20	150	2.40	3600.00	2.82	0.86	6.37	1.16	247.92
21	25	60	1.80	3600.00	2.42	1.12	4.49	0.86	67.32
22	25	70	3.51	3600.00	5.73	4.25	6.94	0.66	87.83
23	25	80	3.30	3600.00	1.57	0.09	4.84	1.16	108.34
24	25	90	3.84	3600.00	3.06	0.73	5.30	1.26	96.87
25	25	100	2.78	3600.00	4.21	2.86	5.76	0.73	153.15
26	25	110	1.75	3600.00	4.81	3.45	7.61	0.98	158.15
27	25	120	2.95	3600.00	4.30	2.51	6.53	0.91	197.62
28	25	130	2.51	3600.00	5.55	3.58	9.23	1.07	232.82
29	25	140	2.65	3600.00	1.96	0.06	5.04	1.16	292.23
30	25	150	1.41	3600.00	3.20	1.01	7.40	1.34	345.69
31	30	60	2.52	3600.00	0.97	0.42	2.80	0.49	33.63
32	30	70	3.53	3600.00	5.87	3.83	6.98	0.73	65.26
33	30	80	2.01	3600.00	1.65	0.34	3.46	0.79	116.14
34	30	90	0.56	3600.00	1.68	0.00	4.42	1.30	210.71
35	30	100	1.25	3600.00	2.85	0.80	5.91	1.32	253.13
36	30	110	2.28	3600.00	3.11	1.26	5.04	0.95	162.67
37	30	120	2.71	3600.00	4.84	3.31	7.85	7.97	28.95
38	30	130	3.29	3600.00	6.48	4.83	8.16	0.81	318.92
39	30	140	2.73	3600.00	5.69	3.70	7.72	0.80	296.87
40	30	150	2.49	3600.00	4.00	2.13	6.63	1.17	331.43
Average			2.68	3600.00	3.96	2.25	6.30	1.13	153.05

within 30 runs. Further, even the average solutions delivered by the TS show smaller gaps than the solutions of the B & B method for a number of instances. What is more, the near-zero Gap_{sd}s and small Gap_ws for all instances (only Gap_w for instance 19 goes larger than 10%) indicate that our proposed TS has excellent robustness for solving these instances.

7.4.2. Results of instances in Part E

The last part of our test focuses on Part E which includes 30 largest instances with 30 to 50 machines and 250 to 500 tasks. Table 7 reports the solutions of the B & B algorithm and the TS for solving these instances.

Table 7: Results of large-sized instances in Part E.

Instance No.	m	n	B & B		TS				
			Gap(%)	Time(s)	Gap_a(%)	Gap_b(%)	Gap_w(%)	Gap_{sd}(%)	Time(s)
1	30	250	12.44	3600.00	7.07	5.22	10.19	1.25	117.51
2	30	300	23.85	3600.00	8.04	5.58	13.52	1.86	204.34
3	30	350	24.70	3600.00	9.47	6.96	19.28	2.32	269.28
4	30	400	23.62	3600.00	7.41	5.27	15.13	1.76	344.61
5	30	450	25.01	3600.00	10.42	7.29	15.21	1.78	417.86
6	30	500	23.53	3600.00	8.78	6.43	11.22	0.97	644.03
7	35	250	26.99	3600.00	8.94	7.55	10.92	0.88	150.66
8	35	300	25.00	3600.00	7.30	5.45	10.55	0.99	259.27
9	35	350	27.19	3600.00	9.72	7.35	14.58	1.50	338.96
10	35	400	32.19	3600.00	9.33	7.97	11.25	0.81	485.64
11	35	450	26.18	3600.00	9.06	6.94	12.47	1.48	709.35
12	35	500	28.90	3600.00	9.57	7.60	14.53	1.50	763.48
13	40	250	25.39	3600.00	9.27	6.51	13.95	1.95	299.90
14	40	300	19.07	3600.00	7.47	5.48	12.55	1.50	377.99
15	40	350	22.19	3600.00	5.33	3.45	11.23	1.48	747.92
16	40	400	23.82	3600.00	9.44	7.64	12.53	1.23	914.16
17	40	450	25.87	3600.00	9.94	7.93	13.10	1.50	1179.37
18	40	500	27.13	3600.00	10.15	8.15	16.54	1.78	1698.74
19	45	250	21.37	3600.00	7.44	5.18	12.18	1.66	417.89
20	45	300	24.75	3600.00	6.56	5.12	8.53	1.04	649.61
21	45	350	23.71	3600.00	8.87	7.42	11.29	0.95	545.00
22	45	400	23.13	3600.00	8.28	6.20	11.95	1.34	801.98
23	45	450	26.18	3600.00	9.27	6.88	12.54	1.27	991.76
24	45	500	23.72	3600.00	8.48	6.99	10.17	0.80	1277.15
25	50	250	23.43	3600.00	7.62	5.65	10.94	0.96	312.06
26	50	300	25.23	3600.00	9.77	7.28	12.51	1.12	469.40
27	50	350	23.93	3600.00	10.26	8.24	14.88	1.42	608.39
28	50	400	26.94	3600.00	10.03	8.28	11.89	0.93	932.50
29	50	450	24.34	3600.00	9.08	7.32	13.65	1.35	2157.38
30	50	500	21.32	3600.00	6.30	3.86	9.25	1.35	2343.65
Average			24.37	3600.00	8.62	6.57	12.62	1.36	714.33

As is shown in Table 7, the B & B method fails to well solve instances with such sizes. The algorithm reaches 3600 s for every instance and the solution gaps are all larger than 10%. In comparison, the TS can better solve instances in this part. The average Gap_a is less than 9% and the average Gap_b is merely 6.57%. In addition, even the Gap_ws are better than the Gaps delivered by the B & B method for all instances. Besides, the Gap_{sd}s are small for all of the instances, indicating the algorithm has very good robustness. Moreover, the solution time shows that the TS is able to solve all instances in a reasonable time.

7.5. *Performances of lower bounds and fathoming tests in the B & B method*

In this section, we give an analysis of the performances of lower bounds and fathoming tests in the B & B method proposed in Section 5.3.2 and Section 5.2.1. For each newly generated node in the B & B procedure, it can be deleted because (1) the node is dominated by other nodes according to Property 5.1 (denoted by $FT1$), Property 5.2 (denoted by $FT2$) or Property 5.3 (denoted by $FT3$) or because (2) the node is dominated by the current upper bound as its lower bound is calculated to be no less than the current solution, otherwise, (3) the node will be retained and assigned with a lower bound equal to $\max_{i\in\{1,2,3,4,5\}} LB_i$.

For any given instance we can easily obtain among all the generated nodes the percentages that are deleted due to $FT1$, $FT2$, or $FT3$, and for each LB_i, the percentage that is either deleted because LB_i is larger than the upper bound or assigned with LB_i as the lower bound. We record such percentages for all instances and report them in average values of different groups of instances (instances in the same Part with the same number of machines are tackled as a group). Table 8 demonstrates the results.

Table 8: Performances of lower bounds and fathoming tests

Part	m	LB1(%)	LB2(%)	LB3(%)	LB4(%)	LB5(%)	FT1(%)	FT2(%)	FT3(%)
A	3	12.15	29.72	5.66	0.00	0.69	8.86	1.49	41.43
A	4	6.51	19.36	5.28	11.00	0.26	9.55	1.69	46.35
A	5	3.51	20.38	4.64	3.44	0.61	15.51	0.42	51.49
B	3	9.88	13.35	3.42	0.00	0.18	18.46	4.37	50.34
B	4	7.73	5.25	1.24	14.70	0.06	30.64	5.59	34.78
B	5	4.18	9.48	1.63	5.13	0.04	37.46	5.24	36.84
C	6	9.89	7.13	0.36	38.96	0.01	25.57	1.68	16.42
C	8	8.43	5.75	0.53	40.56	0.01	26.51	1.57	16.64
C	10	6.24	5.71	0.20	40.07	0.01	30.74	1.00	16.03
D	15	16.54	7.49	1.94	29.22	0.04	24.62	1.59	18.56
D	20	11.48	5.40	0.39	41.67	0.01	22.73	1.39	16.94
D	25	5.12	5.70	0.12	43.05	0.00	29.22	1.52	15.28
D	30	4.10	6.23	0.16	44.85	0.01	25.03	1.63	17.99
E	30	10.25	0.49	0.01	67.31	0.00	13.12	1.16	7.67
E	35	1.73	0.28	0.00	79.32	0.00	9.15	1.65	7.87
E	40	0.00	3.70	0.00	77.43	0.00	10.63	1.38	6.86
E	45	0.00	13.05	0.00	66.64	0.00	12.04	0.96	7.31
E	50	0.00	0.06	0.00	80.15	0.00	11.74	1.27	6.79
Average		6.54	8.81	1.42	37.97	0.11	20.09	1.98	23.09

As is shown in the table, when $m = 3$, LB_4 fails to provide valid lower bounds for all instances. This verifies our observation proposed in Section 5.3.2. In addition, the table shows that quite a proportion of nodes in any instance are deleted by $FT1$, $FT2$, and $FT3$ and this proportion first increases and then decreases with the increase of problem scale. In addition, when the problem scale gets larger, the strengths of $LB2$, $LB3$ and $FT3$ generally decrease while the strength of $LB4$ increases gradually. Generally speaking, $LB4$ provides the tightest lower bound for the $P|size_j|C_{\max}$ problem, followed by $LB2$, $LB1$ and then $LB3$, and $LB5$ seems to be relatively less effective.

8. Conclusion

This work analyzes the $P|size_j|C_{\max}$ problem. We provide several properties for an optimal solution of the problem and formulated it in two different mathematical formulations. The two formulations can be solved by CPLEX on small instances.

To solve large instances efficiently, an exact B & B method with 3 fathoming criteria and 5 lower bounds and a heuristic TS algorithm are developed. A series of computational experiments are then conducted to examine the efficiencies of these methods. The results show that the B & B method performs very well for problem instances with up to 5 machines and 20 tasks, and it can also obtain optimal or near-optimal solutions within a reasonable time for instances with up to 30 machines and 150 tasks. Meanwhile, the TS can better solve large scaled instances and is also proved to be capable of obtaining high-quality solutions for the $P|size_j|C_{\max}$ problem with different sizes in a robust and efficient way.

For future studies, we find two promising directions. First, it would be an interesting topic to further improve the B & B method and the TS or to develop new exact or approximate algorithms which can solve the $P|size_j|C_{\max}$ problem more efficiently. Second, there are a lot of variants of the $P|size_j|C_{\max}$ problem where the machine environment, optimizing objective or other conditions are different from the current problem, and studies of such problems are natural extensions of the current work.

References

Amoura, Bampis, Kenyon, Manoussakis, 2002. Scheduling independent multiprocessor tasks. Algorithmica 32, 247–261.

Blazewicz, J., DellOlmo, P., Drozdowski, M., Mczka, P., 2003. Scheduling multiprocessor tasks on parallel processors with limited availability. European Journal of Operational Research 149, 377–389.

Blazewicz, J., Drabowski, M., Weglarz, J., 1986. Scheduling multiprocessor tasks to minimize schedule length. IEEE Transactions on Computers 35, 389–393.

Blazewicz, J., Lenstra, J.K., Kan, A.R., 1983. Scheduling subject to resource constraints: classification and complexity. Discrete Applied Mathematics 5, 11–24.

Blkadek, I., Drozdowski, M., Guinand, F., Schepler, X., 2015. On contiguous and non-contiguous parallel task scheduling. Journal of Scheduling 18, 487–495.

Brucker, P., 2006. Multiprocessor tasks, in: Scheduling Algorithms. Springer, pp. 317–342.

Chen, J., Lee, C.Y., 1999. General multiprocessor task scheduling. Naval Research Logistics 46, 57–74.

Chou, F.D., 2013. Particle swarm optimization with cocktail decoding method for hybrid flow shop scheduling problems with multiprocessor tasks. International Journal of Production Economics 141, 137–145.

Cordeau, J., Gendreau, M., Laporte, G., 1997. A tabu search heuristic for periodic and multidepot vehicle routing problems. Networks 30, 105–119.

Drozdowski, M., 1996. Scheduling multiprocessor tasks - an overview. European Journal of Operational Research 94, 215–230.

Du, J., Leung, J.Y.T., 1989. Complexity of scheduling parallel task systems. SIAM Journal on Discrete Mathematics 2, 473–483.

França, P.M., Gendreau, M., Laporte, G., Müller, F.M., 1996. A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. International Journal of Production Economics 43, 79–89.

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 2, 533–549.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.R., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey, in: Annals of Discrete Mathematics. Elsevier. volume 5, pp. 287–326.

Hidri, L., 2016. Note on the hybrid flowshop scheduling problem with multiprocessor tasks. International Journal of Production Economics 182, 531–534.

Johannes, B., 2006. Scheduling parallel jobs to minimize the makespan. Journal of Scheduling 9, 433–452.

Krawczyk, H., Kubale, M., 1985. An approximation algorithm for diagnostic test scheduling in multicomputer systems. IEEE Transactions on Computers 34, 869–872.

Lee, C.Y., 1991. Parallel machines scheduling with nonsimultaneous machine available time. Discrete Applied Mathematics 30, 53–61.

Lee, C.Y., Lei, L., Pinedo, M., 1997. Current trends in deterministic scheduling. Annals of Operations Research 70, 1–41.

Leung, J.Y., 2004. Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press.

Li, K., 1999. Analysis of an approximation algorithm for scheduling independent parallel tasks. Discrete Mathematics and Theoretical Computer Science 3, 155–166.

Lin, J.F., Chen, S.J., 1994. Scheduling algorithm for nonpreemptive multiprocessor tasks. Computers and Mathematics with Applications 28, 85–92.

Lloyd, E.L., 1981. Concurrent task systems. Operations Research 29, 189–201.

Mensendiek, A., Gupta, J.N.D., Herrmann, J., 2015. Scheduling identical parallel machines with fixed delivery dates to minimize total tardiness. European Journal of Operational Research 243, 514–522.

Roy, B., Sussmann, B., 1964. Les problmes dordonnancement avec contraintes disjonctives. Note ds 9.

Sammarra, M., Cordeau, J.F., Laporte, G., Monaco, M.F., 2007. A tabu search heuristic for the quay crane scheduling problem. Journal of Scheduling 10, 327–336.

Schutten, J., 1996. List scheduling revisited. Operations Research Letters 18, 167–170.

Solimanpur, M., Elmi, A., 2013. A tabu search approach for cell scheduling problem with makespan criterion. International Journal of Production Economics 141, 639–645.

Switalski, P., Seredynski, F., 2015. Scheduling parallel batch jobs in grids with evolutionary metaheuristics. Journal of Scheduling 18, 345–357.

Trkoullar, Y.B., TaÅkn, Z.C., Aras, N., Altnel, .K., 2014. Optimal berth allocation and time-invariant quay crane assignment in container terminals. European Journal of Operational Research 235, 88–101.
Webster, S.T., 1996. A general lower bound for the makespan problem. European Journal of Operational Research 89, 516–524.