Research Article

A Column Generation Algorithm for the Resource-Constrained Order Acceptance and Scheduling on Unrelated Parallel Machines

Yujian Song , Ming Xue , Changhua Hua , and Wanli Wang

School of Business Administration, Shanghai Lixin University of Accounting and Finance, Shanghai 201620, China

Correspondence should be addressed to Wanli Wang; 20090898@lixin.edu.cn

Received 22 February 2021; Revised 7 September 2021; Accepted 17 September 2021; Published 20 October 2021

Academic Editor: Adrian Neagu

Copyright © 2021 Yujian Song et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In this paper, we investigate the resource-constrained order acceptance and scheduling on unrelated parallel machines that arise in make-to-order systems. The objective of this problem is to simultaneously select a subset of orders to be processed and schedule the accepted orders on unrelated machines in such a way that the resources are not overutilized at any time. We first propose two formulations for the problem: mixed integer linear programming formulation and set partitioning. In view of the complexity of the problem, we then develop a column generation approach based on the set partitioning formulation. In the proposed column generation approach, a differential evolution algorithm is designed to solve subproblems efficiently. Extensive numerical experiments on different-sized instances are conducted, and the results demonstrate that the proposed column generation algorithm reports optimal or near-optimal solutions that are evidently better than the solutions obtained by solving the mixed integer linear programming formulation.

1. Introduction

In many industries, the increasing demand for product variety and customization has intensified competitions among manufacturers to such an extent that attracting and retaining customers is more important than before. The fierce competition has stimulated manufacturers to switch from make-to-stock production to make-to-order production. In the make-to-order production system, manufacturing of the product begins after receiving a customer order. Undoubtedly, accepting all potential orders may cause overloads, order delays, and customer dissatisfaction, due to the limited production capacity and tight delivery requirements. Therefore, the first and foremost decision for a make-to-order system is to decide simultaneously which orders to accept and how to schedule the accepted orders.

Due to its application in diverse industries, the order acceptance and scheduling problem has drawn considerable attention from academics as well as practitioners. It has been considered in single machine, parallel machine, and flow shop environments. Various formulations and solution

methods for this problem have been proposed. Although there is abundant literature on the order acceptance and scheduling problem, most of these studies treat machine as the only required resource. However, in real-world make-to-order production systems, other resources such as machine operators, tools, fixtures, or industrial robots are also needed and restricted in quantity. An example can be found in the plastic injection molding industry. Plastic injection molding is a production method used to produce plastic products by injecting heated plastic into a preformed mold. The molten plastic is injected at a predetermined temperature and pressure and left to cool. When it is solidified and adapted the shape of the mold, it is ejected as a formed product and the molding cycle is repeated. Figure 1 illustrates the process of plastic injection molding. With no doubt, the injection machine can only start its processing when the mold suitable for the products to be produced is mounted on the injection machine. In general, the molds are quite expensive, and thus the number of molds available is limited. It is obvious that when a manufacturing system like this has limited additional resources besides machines, and thus is unable to meet all customers' demand, it has to decide which orders are to be

FIGURE 1: The process of plastic injection molding (<http://www.aireplatstic.com>).

accepted for processing as well as their associated schedule. Therefore, it is of great practical significance to investigate the resource-constrained order acceptance and scheduling problem.

In this paper, we try to fill the gap and address the resource-constrained order acceptance and scheduling problem on unrelated parallel machines. The objective of this problem is to simultaneously select a subset of orders to be processed and schedule the accepted orders on unrelated machines so as to ensure that the resources are not overutilized at any time. The main contributions of this research are threefold.

- (1) We consider a new variant of the order acceptance and scheduling problem on unrelated machines, which takes the additional limited resources into consideration. We present an integer programming model and a set partitioning formulation for this problem.
- (2) We develop a column generation algorithm based on the set partitioning formulation. In order to solve the pricing subproblems efficiently, we devise a differential evolution algorithm to rapidly price out promising columns.
- (3) We conduct comprehensive numerical experiments on randomly generated instances to illustrate the efficacy of the proposed column generation algorithm.

The remainder of the paper is organized as follows. Section 2 reviews relevant literature. Problem definition and the integer linear programming model are provided in Section 3, followed by a description of the proposed column generation algorithm and its implementation in Section 4. In

Section 5, we discuss our numerical results and validate the effectiveness and the efficacy of the proposed solution approach. Section 6 concludes the contribution and recommends directions for future research.

2. Literature Review

The scheduling problem addressed in this work is related to two fields of research: (i) order acceptance and scheduling and (ii) resource-constrained unrelated parallel machine scheduling. Existing studies on both topics is reviewed in this section.

2.1. The Order Acceptance and Scheduling on Unrelated Parallel Machines. Due to its wide applications in the real world, the order acceptance and scheduling problem and its variants have been extensively studied. Interested readers can read Slotnick's study [1] for a comprehensive literature review of the order acceptance and scheduling problem. The following survey only includes a representative subset of papers addressing the order acceptance and scheduling problem on unrelated parallel machines.

Fanjul-Peyro and Ruiz [2] investigated two generalizations of the unrelated parallel machine scheduling problem, one of which is concerned with the integrated order selection and scheduling decision making problem. A mixed integer programming mathematical formulation is presented, along with a constructive method specifically tailored for such a generalization. Comprehensive numerical and statistical experiments show that the proposed method significantly improves the results yielded by the CPLEX solver. Hsu and Chang [3] dedicatedly study the unrelated parallel-machine scheduling problem

with deteriorating jobs and rejection. The objective of this problem is to reject a subset of jobs and then schedule the accepted jobs in such a way to minimize the total cost. Computational experiments show that the problem can be solved in polynomial time when the quantity of machines is fixed. Jiang and Tan [4] study the unrelated parallel machine scheduling model with job acceptance option. Each job order is either accepted and processed without interruption, or is declined at a penalty cost. In an attempt to minimize the makespan of all accepted jobs, the penalty cost of rejected jobs, and the production cost of accepted jobs, the authors first develop a polynomial time algorithm to derive a lower bound and then propose a heuristic by introducing various relaxation techniques. The worst-case ratio bound of the proposed approach is proven to be 2.

Emami et al. [5] consider the simultaneous order acceptance and scheduling problem in an unrelated parallel machines environment with the assumption of uncertain revenue and job-processing times. The objective of this problem is to find a subset of orders and schedule them in such a way that the total profit is maximized. A Lagrangian relaxation algorithm enhanced by a cutting plane technique is implemented and capable of solving instances with up to 40 orders and 6 machines. The same problem is studied in Emami et al. [6] and solved by Benders decomposition approach. Wang and Wang [7] mainly focus on the bi-objective optimization of order acceptance and scheduling on unrelated parallel machines with machine eligibility constraints, where a subset of jobs has to be selected to simultaneously maximize the profit level and to minimize the total completion time. The problem is formulated as a biobjective mixed integer linear programming. In view of its NP-hardness, a multiobjective parthenogenetic algorithm with parthenogenetic operators and Pareto-ranking and selection methods are implemented to obtain approximate Pareto front. Mor and Mosheiov [8] study the order acceptance and scheduling problem on unrelated parallel machines, which takes the position-dependent processing times into consideration. This problem aims at minimizing total absolute deviation of job completion times. It is shown that the problem is solved in polynomial time in the number of jobs, and a polynomial time solution approach based on solving a sequence of linear assignment problems is designed to solve this problem. Wang and Ye [9] investigate the order acceptance and scheduling problem on unrelated parallel machines. The purpose is to determine the orders to be accepted for processing and the processing sequence for the accepted orders to get the optimal profit. Two mixed integer programming formulations are presented, which are further enhanced by various valid inequalities. For the purpose of solving complicated instances, the authors propose a tailored branch-and-bound algorithm. The proposed branch-and-bound algorithm which follows the paradigm of “divide and conquer” branches on the number of accepted orders and recursively solves the reduced unrelated parallel machine scheduling problem at each child node. Computational experiments confirm its effectiveness.

2.2. The Resource-Constrained Parallel Machine Scheduling. Despite the plentiful achievements mentioned above, research on order acceptance and scheduling with explicit consideration of additional constrained resources is somewhat limited. Since the considered problem is closely related to the resource-constrained parallel machine scheduling, an insight into its recent advances has substantial implications in mathematical modeling and solution development for the studied problem. Therefore, a brief literature review on it is presented as follows.

Ventura and Kim [10] investigate a resource-constrained parallel machine scheduling problem which arises in the burn-in process of semiconductor manufacturing. The problem is formulated as an integer programming model with the objective of minimizing the total absolute deviations of job completion times from due dates. The problem is proved to be solved in polynomial time when there is one single type of additional resource, and each job consumes one unit of additional resource at most. Ventura and Kim [11] extend the work of [10] by considering that the number of kinds of additional resources and the resource requirements per job are unrestricted. This more general problem is modelled as a 0-1 integer program, and the Lagrangian relaxation algorithm is applied to obtain high-quality solutions and tight lower bounds for large-scale problems. Chen [12] studies the resource-constrained unrelated parallel machine scheduling problem which originates from the job scheduling in the plastics-forming industry. Beside plastic moulding machines, scarce and costly dyes are also indispensable equipment and thus treated as secondary resource constraints for plastic forming operations. A heuristic is developed to minimize the makespan. The computational results show that it outperforms the simulation annealing algorithm in terms of efficiency and effectiveness. This work is further improved by Chen and Wu [13], who propose an effective tabu search algorithm. Ruiz-Torres et al. [14] investigate the problem of scheduling jobs on uniform parallel machines, where the speed of the machine is determined by the quantity of the assigned additional resource. The objective aim is to minimize the total number of delayed jobs. Two different versions of the problem are discussed. The first one makes the assumption that all jobs are preallocated to the machines, while the second one considers simultaneous optimization of job scheduling and resource allocation. An integer programming formulation is presented for the first case and five heuristics for the second. Edis et al. [15] comprehensively survey the related literature on resource-constrained parallel machine scheduling. They classify the studies with respect to the characteristics of machine environments, objective functions, additional resource types, complexity of the problem and solution approaches, and other important issues. Strengths and limitations of the existing literature as well as open areas for future studies are also highlighted in this paper. Afzalirad and Rezaeian [16] deal with a resource-constrained unrelated parallel machine scheduling problem which considers various realistic features, such as release dates, sequence-

dependent setup times, machine eligibility, and precedence restrictions. An integer programming model is proposed to represent the complicated problem mathematically. Two different heuristic algorithms including genetic algorithm and artificial immune algorithm are designed to identify high-quality solutions. Comparative experiments are performed on a set of randomly generated test instances and demonstrate that both algorithms perform rather well for small-scale instances and the artificial immune algorithm is suggested to solve large-scale instances. Li et al. [17] delicately address a job-scheduling problem arising in the manufacturing systems exposed to both machine and worker availability constraints. A branch population genetic algorithm is proposed to minimize the makespan and cost. The effectiveness of the proposed algorithm is validated by comparative experiments and an actual case study, with the conclusion that the proposed algorithm is very useful under the realistic conditions. Fanjul-Peyro et al. [18] investigate a resource-constrained parallel machine scheduling problem where the processing of jobs on the machines demands a number of units of a scarce resource. Two mixed integer linear programming (MILP) models are presented to solve small-sized instances. Besides, three meta-heuristic algorithms based on the MILP models are developed to solve large-sized instances. Vallada et al. [19] mainly focus on the solution development for the resource-constrained unrelated parallel machine scheduling problem. They propose a basic scatter search algorithm, an enriched scatter search algorithm and an enriched iterated greedy algorithm, to solve this problem. In all the proposed methods, nonfeasible solutions with violation of resource limitation are allowed and further fixed to a feasible solution using a repairing mechanism. Different local search procedures based on insertion, swap, and restricted neighborhoods are implemented to enhance the performance of the proposed approaches. Nguyen et al. [20] deal with a resource-constrained job scheduling problem abstracted from mineral transportation. A parallel differential evolution algorithm is implemented in conjunction with iterated greedy search and column generation improvement strategies to solve this problem. They conduct an exhaustive numerical evaluation of the proposed and existing methods using different-sized instances. The computational results confirm the efficiency and effectiveness of the proposed algorithm and advantage of the adopted improvement strategies. More recently, Yepes-Borrero et al. [21] address the unrelated parallel machine scheduling problem with setup times and additional resources in the setups. A MILP formulation and three meta-heuristics are proposed for this problem.

In summary, there are a number of studies on order acceptance and scheduling as well as resource-constrained parallel machine scheduling, but this field of research still requires further development due to its widespread applications. The key extension of our paper which differentiates ours from existing studies is the consideration of order acceptance and resource-constrained parallel machine scheduling in an integrated manner.

3. Problem Definition and Mathematical Model

In this section, we mainly elaborate on the problem statement and provide an integer programming formulation.

3.1. Problem Statement and Integer Programming Formulation. This paper considers a setting where a manufacture receives orders from its customers but is confronted by additional limited resources. Specifically, let $M = \{1, \dots, |M|\}$ be the set of unrelated machines and $N = \{1, \dots, |N|\}$ be the set of orders. The maximum amount of resource k $\{k \in K\}$ is constrained to be $Q_{\max}^k$. Each machine can process only one order at any time and is always available within the planning horizon $T = \{0, 1, \dots, |T|\}$. Each order can be processed on atmost one machine without interruption and requires a constant number of the additional resources during its entire process. Thus, each order can only be processed when one machine and sufficient additional resources are available during its processing time.

Associated with order i are as follows: (1) release date r_i ; (2) processing time p_{im} , corresponding to the time required to process order i on machine m ; (3) due date d_i ; (4) resource requirement of order i to resource k , q_i^k ; (5) revenue e_i gained from its customer if it is accepted; and (6) penalty w_i for each unit time of tardiness. Since tardiness penalties will incur a loss of revenue, the objective of this problem is to select a subset of orders and schedule them in such a way that all constraints are satisfied and the total net revenue is maximized. To formulate the problem, the following notations are defined.

Parameters:

- T : set of time periods in the scheduling horizon, $T = \{0, 1, \dots, |T|\}$
- M : set of unrelated machines, $M = \{1, \dots, |M|\}$
- N : set of orders, $N = \{1, \dots, |N|\}$
- K : set of resource types, $N = \{1, \dots, |K|\}$
- r_i : release time of order i
- p_{im} : processing time of order i on machine m
- d_i : due time of order i
- q_i^k : the amount of resource k required for the processing of order i
- e_i : revenue of order i
- w_i : unit penalty cost of order i
- $Q_{\max}^k$: maximum amount of the additional limited resource k

Decision variables:

- X_i : a binary variable, equal to 1 if order i is accepted and 0 otherwise
- Y_{itm} : a binary variable, equal to 1 if order i completes its processing at time t on machine m and 0 otherwise
- C_i : completion time of order i
- T_i : tardiness of order i

The problem described above can be mathematically formulated as the following MILP model.

$$\max \sum_{i \in N} (e_i * X_i - w_i * T_i), \quad (1)$$

$$\text{s.t.} \sum_{i \in N} \sum_{s \in \{t, t+p_{im}-1\}} Y_{ism} \leq 1, \quad \forall m \in M, t \in T, \quad (2)$$

$$\sum_{m \in M} \sum_{t \in \{r_i+p_{im}, T\}} Y_{itm} \leq 1, \quad \forall i \in N, \quad (3)$$

$$\sum_{m \in M} \sum_{i \in N} \sum_{s \in [t, t+p_{im}-1]} q_i^k * Y_{ism} \leq Q_{\max}^k, \quad \forall t \in T, k \in K, \quad (4)$$

$$\sum_{m \in M} \sum_{t \in \{0, r_i+p_{im}-1\}} Y_{itm} = 0, \quad \forall i \in N, \quad (5)$$

$$\sum_{m \in M} \sum_{t \in T} t * Y_{itm} = C_i, \quad \forall i \in N, \quad (6)$$

$$\sum_{m \in M} \sum_{t \in T} Y_{itm} = X_i, \quad \forall i \in N, \quad (7)$$

$$T_i \geq C_i - d_i, \quad \forall i \in N, \quad (8)$$

$$C_i, T_i \geq 0, X_i, S_{itm} \in \{0, 1\}, \quad \forall i \in N, t \in T, m \in M. \quad (9)$$

The objective function (1) seeks to maximize the total net revenue of accepted orders, which is computed by the total revenue minus the total tardiness penalty. Constraint (2) ensures that not more than one order can be assigned to any machine at any time period. Constraint (3) specifies that an order is either accepted or rejected. Constraint (4) restricts that the total amount of the additional resource k consumed at any time should not exceed the available amount $Q_{\max}^k$. Constraint (5) restricts that an order cannot be processed before its release time. Constraint (6) computes the completion time for each order. Constraint (7) links variable Y_{itm} and variable X_i and is introduced to determine if an order is accepted or not. Constraint (8) calculates the tardiness for each order. Constraint (9) specifies the domains of decision variables.

3.2. An Illustrative Example. To verify the proposed MILP formulation and intuitively understand this complicated scheduling problem, we consider a simple example involving 6 jobs, 2 machines, and one additional resource type. The maximum amount of the additional resource is 6 units. The parameters related to the example are provided in Table 1. The mathematical formulation of this example is solved to optimality using CPLEX 12.10 software. The net revenue of the optimal solution is 48. In Figure 2, the optimal schedule is visualized in the Gantt chart. The horizontal axis of the chart shows the time when each order is under processing, while the vertical axis indicates the name of each machine.

TABLE 1: Example problem data.

Order	r_i	q_i	d_i	e_i	w_i	p_{im}	
						Machine 1	Machine 2
Order 1	1	2	5	10	2	3	2
Order 2	2	3	6	12	3	2	3
Order 3	3	4	6	10	4	3	3
Order 4	3	3	7	9	1	3	1
Order 5	4	4	8	11	2	1	2
Order 6	4	3	9	10	2	6	6

Each block on the chart corresponds to an order, which is marked by its order name and the amount of resource required for processing.

It can be seen that all orders except order 3 are accepted. We can also observe that machine 2 is idle from time 4 to time 5 since order 5 and order 6 cannot be processed at the same time due to limited resources. From this example, we can know that the considered problem is difficult to solve because its solution requires not only selecting a subset of orders but also scheduling them on machines in such a way that the resources are not overused at any time.

3.3. Complexity of the Problem

Theorem 1. *The resource-constrained order acceptance and scheduling on unrelated parallel machines is strongly NP-hard.*

Proof. When we set the maximum amount of the additional resource to sufficiently large and introduce a dummy machine to which all rejected orders are assigned with zero processing time, zero revenue, and zero penalty, the studied problem can be reduced to the classical unrelated parallel machine scheduling problem which is known to be strongly NP-hard (Liaw et al. [22]). Because the classical unrelated parallel machine scheduling problem is a special case of the considered problem, it follows that the considered problem is also strongly NP-hard. $\square$

4. Column Generation Approach

The MILP formulation presented in Section 3 is capable of solving small instances to optimality in reasonable time, but it becomes computationally expensive with the increase of the problem size, as shown in Section 5. This fact necessitates the development of efficient solution methodologies. Column generation is an effective and efficient methodology which has been well studied and successfully applied to a wide range of problems. In this section, we provide an alternative set partitioning formulation and then develop a tailored column generation approach to solve the considered problem.

4.1. An Overview of the Proposed Column Generation Algorithm. Column generation is an efficient approach for solving linear programming problems with a large number of variables (columns). Its main advantage is that not all

FIGURE 2: The optimal solution to the illustrative example. (a) The Gantt chart of the optimal schedule. (b) Resource consumption.

columns need to be enumerated. Instead, the problem is first formulated as a restricted linear master problem (RLMP). This RLMP considers only a subset of columns. And then, one or more subproblems with respect to current dual solution to the RLMP are solved to identify favorable new columns which can be incorporated into the RLMP to improve the objective value. The RLMP and subproblems are solved iteratively until no promising columns can be further added to the RLMP. This termination condition implies that the RLMP is optimal. The RLMP probably returns a fractional solution. In this paper, in order to obtain an integer solution, the RLMP with all columns which have ever been generated as decision variables is converted to an IP model and then solved by an IP solver.

In an attempt to enhance the performance of the column generation, we also propose a differential evolution (DE) algorithm to solve the subproblems efficiently. Besides, we develop a constructive heuristic to obtain the initial solution for the RLMP. The procedures of the proposed column generation approach are shown in Algorithm 1. Next, we present a detailed description of each step.

4.2. Master Problem: A Set Partitioning Formulation. In this section, we reformulate the considered problem as a set partitioning formulation which serves as the master problem in the column generation framework. The set partitioning formulation which is a special class of linear integer programs has been utilized to formulate a wide range of practical problems in fields such as crew scheduling, vehicle routing, and timetable scheduling. Based on the special structure of the set partitioning formulation, efficient approaches techniques, for example, column generation, have been proposed to solve large-sized problems, making it possible to deal with many real-life applications. Therefore, we attempt to propose an alternative mathematical formulation based on the set partitioning formulation.

In the set partitioning formulation, there is a set of columns (assignments) denoted by Ω that composes the set partitioning formulation matrix. Each column corresponds to a feasible assignment of a single order to a machine in accordance with the time-related constraints. This matrix contains three submatrices, denoted as A , B , and C , all containing $|\Omega|$ columns. Matrix $A = (a_{i\omega})$ contains a row for each order, and $a_{i\omega} = 1$, if and only if column ω represents a feasible assignment of order $i \in N$. In matrix $B = (b_{m\omega})$,

$m \in M$; $t \in T$, a single row corresponds to a particular (machine, time) position. If machine m is in use at time t in assignment ω , then $b_{mt\omega} = 1$ and 0 otherwise. In matrix $C = (c_{t\omega}^k)$, a single row corresponds to a time interval, each element of which represents the number of the additional resource assigned in time interval t to process the order which column ω represents. Each column ω is associated with a profit R_ω which is defined as the net revenue of the respective assignment of a specific order to a machine. The profit R_ω of column ω can be easily computed based on its corresponding order index and the (machine, time) position. The decision variable is denoted by x_ω which equals 1 if column ω is used in the solution, and 0, otherwise.

Therefore, the set partitioning formulation is mathematically represented as follows:

$$\max \sum_{\omega \in \Omega} R_\omega x_\omega, \quad (10)$$

$$\text{s.t. } \sum_{\omega \in \Omega} a_{i\omega} x_\omega \leq 1, \quad \forall i \in N, \quad (11)$$

$$\sum_{\omega \in \Omega} b_{mt\omega} x_\omega \leq 1, \quad \forall t \in T, m \in M, \quad (12)$$

$$\sum_{\omega \in \Omega} c_{t\omega}^k x_\omega \leq Q_{\max}^k, \quad \forall t \in T, k \in K, \quad (13)$$

$$x_\omega \in \{0, 1\}, \quad \forall \omega \in \Omega. \quad (14)$$

The objective function (10) is to maximize the sum of revenue of selected assignments. Constraint (11) indicates that any order can be accepted at most once. Constraint (12) limits that a given (machine, time) position can only process at most one order. Constraint (13) ensures that the resources are not overutilized at any time. Constraint (14) defines the domains of variables.

In practice, the number of feasible assignments (columns) grows exponentially with the problem size. Therefore, it is computationally intractable to completely enumerate all feasible columns, making the explicit resolution of the set partitioning formulation impractical for most real-world applications. As an alternative, we solve the set partitioning formulation problem in a column generation framework.

The underlying idea of column generation is not to explicitly list all of the columns of the set partitioning

```

Step 1: implement the proposed constructive heuristic to obtain a feasible solution
Step 2: initialize the RLMP with the solution obtained by the constructive heuristic
Step3: set the number of consecutive iterations of failing to find a column with positive reduced cost  $Iter_{non}$  to be 0, i.e.,  $Iter_{non} = 0$ 
Step 4: while  $Iter_{non}$  is less than a specified number  $Iter_{nonCG}$ , do
Step 4.1: solve the RLMP and get the dual solution
Step 4.2: define the subproblems by the current dual solution and solve them by the proposed DE algorithm
Step 4.3: If there exist columns with positive reduced cost, do
    Add new columns to the RLMP;  $Iter_{non} = 0$ 
Else
     $Iter_{non} = Iter_{non} + 1$ 
End
End
Step 5: convert the RLMP model to an IP model
Step 6: solve the IP model by CPLEX
Step 7: output the solution

```

ALGORITHM 1: The procedure of the proposed column generation.

formulation (10)–(14), but rather to dynamically generate promising columns. To be more specific, column generation starts with a “simple” formulation including a few columns $\bar{\omega} \in \Omega$ and relaxes the integrality restrictions on the variables, i.e., $x_{\bar{\omega}} \leq 0$, which is referred to as a *restricted linear master problem* (RLMP). Subproblems with respect to current dual variables of the RLMP are solved to identify favorable new columns which can be incorporated into the RLMP to improve the objective. The RLMP and subproblems are solved iteratively until no promising columns can be further identified.

4.3. The Initial Solution to the RLMP. As mentioned before, column generation starts with only a small subset of columns and then generates necessary columns by an iterative procedure between the restricted master problem and subproblems. Therefore, we should initialize the RLMP. In this section, we design a constructive heuristic to obtain the initial solution for the RLMP. The heuristic works as follows. In the first step, the most urgent order is selected (i.e., the one with the earliest release time). Then, in the second step, it is tentatively inserted into the end of each machine’s current partial schedule. The order is rejected if its every possible insertion incurs a decrease in the net revenue. Otherwise, the machine which leads to the largest positive growth in the net revenue is selected to process the order, and the order is placed at the end of this machine’s partial schedule. These two steps iterate until all orders have been checked. The

constructive heuristic can always yield a feasible scheduling plan. Afterwards, the scheduling plan is transformed into a series of assignments of orders to machines, which are mathematically represented by columns and construct our initial RLMP.

4.4. Pricing Subproblems and the DE Algorithm

4.4.1. Mathematical Formulation of the Pricing Subproblem. We solve the pricing subproblems to identify promising columns with positive reduced cost, whose inclusion in the restricted linear master problem will improve the objective value. Because machines are assumed to be unrelated, there are $|M|$ pricing subproblems in total, one for each machine. That is, for each machine m , we try to find an assignment ω which has a positive reduced cost calculated by $\bar{C}_{m\omega} = R_{\omega} - \sum_{i \in N} a_{i\omega} \beta_i - \sum_{t \in T} b_{mt\omega} \sigma_{mt} - \sum_{t \in T} c_{t\omega} \gamma_t$ where β_i , σ_{mt} , and γ_{kt} denote the optimal dual solutions associated with constraints (11)–(13), respectively, in the current iteration of the restricted master problem. The net revenue R_{ω} can also be represented in terms of the variables defined in the original integer programming model (1)–(9), i.e., $R_{\omega} = \sum_{i \in N} (e_i * X_i - w_i * T_i)$. Similarly, we have $a_{i\omega} = X_i$, $b_{mt\omega} = \sum_{i \in N} \sum_{t \in [\max\{t-p_{im}+1, 0\}, t]} S_{itm} \sigma_{mt}$, and $c_{t\omega}^k = \sum_{i \in N} \sum_{t \in [\max\{t-p_{im}+1, 0\}, t]} q_i^k * S_{itm} \gamma_{kt}$, where S_{itm} is equal to 1 if order i starts its processing at time t on machine m and 0 otherwise. Therefore, the pricing subproblem for machine m can be mathematically formulated as follows:

$$\begin{aligned}
 \max \quad & \sum_{i \in N} (e_i * X_i - w_i * T_i) - \sum_{i \in N} X_i \beta_i - \sum_{t \in T} \sum_{i \in N} \sum_{\bar{t} \in [\max\{t-p_{im}+1, 0\}, t]} S_{itm} \sigma_{mt} \\
 & - \sum_{t \in T} \sum_{k \in K} \sum_{i \in N} \sum_{\bar{t} \in [\max\{t-p_{im}+1, 0\}, t]} q_i^k * S_{itm} \gamma_{kt}, \\
 \text{s.t.} \quad & \sum_{i \in N} X_i \leq 1, \quad \forall i \in N,
 \end{aligned} \tag{15}$$

$$\text{s.t.} \quad \sum_{i \in N} X_i \leq 1, \quad \forall i \in N, \tag{16}$$

$$\sum_{t \in T} S_{itm} = X_i, \quad \forall i \in N, \quad (17)$$

$$S_{itm} = 0, \quad \forall i \in N, t < r, \quad (18)$$

$$\sum_{t \in T} (t + p_{im}) * S_{itm} = C_i, \quad \forall i \in N, \quad (19)$$

$$T_i \geq C_i - d_i, \quad \forall i \in N, \quad (20)$$

$$C_i, T_i \geq 0, \quad \forall i \in N, \quad (21)$$

$$X_i, S_{itm} \in \{0, 1\}, \quad \forall i \in N, t \in T. \quad (22)$$

In this model, the objective function (15) maximizes the reduced costs. Constraint (16) ensures that a column is associated with at most one order. Constraint (17) ensures that an order can start its processing at time t on the given machine m only if the order is assigned to machine m . Constraint (18) guarantees that no order is processed before its release time. Constraint (19) computes the completion time of each order. Constraint (20) calculates the tardiness of each order. Constraints (21) and (22) define the regions of decision variables.

Ideally, the solution approach for solving the subproblems should be efficient because it has to be executed many times during the column generation procedure. Undoubtedly, the subproblems of large-sized instances will become computationally challenging for generic state-of-the-art solvers, such as CPLEX. Alternatively, we next introduce a DE algorithm for the pricing subproblems, which can solve the subproblems more efficiently.

4.4.2. DE Algorithm for Pricing Subproblems. The DE algorithm is a simple yet effective metaheuristic algorithm which was first introduced by Storn and Price [23]. Similar to conventional evolutionary algorithms, DE maintains and evolves a population. Specifically, DE starts with a randomly initialized population and then iteratively proceeds from the current population to a new better-performing one via three evolutionary operators, including mutation, crossover, and selection operators. Mutation operator is responsible for generating new mutant individuals, while crossover operator is responsible for exchanging some subcomponents between parent and mutant individuals to produce new trial individuals. Based on a fitness metric, the selection operator compares the parent individuals with the trial ones and the better ones are admitted to the next generation. The three operators iterate until a termination condition is satisfied. The basic evolution process of the DE algorithm is shown in Algorithm 2.

Solution Representation. In order to apply the DE algorithm to solve the subproblems, we need to develop an appropriate encoding approach to represent a feasible solution of a given subproblem. As stated in Section 4.3, the purpose of solving

the subproblems is to identify new columns with positive reduced cost, and each column corresponds to a feasible assignment of a single order to a machine in accordance with the time-related constraints. In view of this, we represent each individual of the population by a vector, whose length is $1 + |N|$. The first element represents the candidate order, which takes value within $[1, |N|]$. The remaining $|N|$ elements corresponds to the starting times of $|N|$ orders, each of which is sampled from its release time to the latest starting time. To better illustrate the solution representation scheme, a simple example for the case mentioned in Section 3.2 is presented in Figure 3.

In this example, the candidate order is order 3, and its starting time is 4. Suppose that the dual variable is given and we are solving the subproblem for machine 1, we can easily obtain a feasible assignment of order 3 to machine 1 and then compute its corresponding reduced cost (the objective function of this individual).

Initialization. The DE population maintains a population of NP vectors. The initial population is randomly generated in the feasible solution domain. For example, the i -th individual can be initialized as follows:

$$X_i^0 = LB + rand * (UB - LB), \quad (23)$$

where $rand$ is a random number that is sampled from a uniform distribution within $(0,1)$. LB and UB is the lower bound vector and upper bound vector of all elements, respectively.

Mutant Operator. At each generation, the mutation operator creates a new mutant individual V_i^t with respect to each individual (target individual $X_{r_1}^t$) of the current population by the mutation operator. There are various mutation strategies. We adopt the DE/rand/1 scheme, which is expressed as follows:

$$V_i^t = X_{r_1}^t + F * (X_{r_2}^t - X_{r_3}^t), \quad (24)$$

where $X_{r_1}^t$, $X_{r_2}^t$, and $X_{r_3}^t$ are mutually different individuals randomly selected from the current population. The scaling factor F controls the mutation scale and is generally distributed within $[0, 1]$.

Step 1: initialize a population of NP individuals
 Step 2: evaluate the fitness values of the NP individuals
 Step 3: **While** the preset stopping condition is not satisfied **DO**
 Step 3.1: Mutation—create a mutant population using a mutation operator
 Step 3.2: Crossover—recombine the parent and the mutant to generate a trial population
 Step 3.3: Selection—compare the parent and trial individuals and select the better ones for the next generation
 End
 Step 4: output the best solution

ALGORITHM 2: The procedure of the DE algorithm.

-	Order 1	Order 2	Order 3	Order 4	Order 5	Order 6	Order ID
3	1	3	4	4	5	7	Value
Candidate order	Starting time						Sub-sequence

FIGURE 3: An example for solution representation.

If the components of the obtained mutant individual exceed the corresponding lower or upper bounds, they can be reinitialized randomly and uniformly within the given range. Since F is a fractional number, V_i^t may also contain

fractional elements. Therefore, if it happens, each fractional element in V_i^t is rounded to its nearest integer.

Crossover Operator. After mutation, in order to increase the diversity of the population, each pair of target individual X_i^t and its corresponding mutant individual V_i^t is crossed to produce a new trial individual U_i^t . In the DE algorithm, the most commonly used crossover strategy is the binomial crossover scheme, which is defined as follows:

$$U_{ij,t} = \begin{cases} V_{ij,t}, & \text{if } rand_{ij} \leq CR \text{ or } j = j_{rand}, \\ X_{ij,t}, & \text{otherwise,} \end{cases} \quad j = \{1, 2, \dots, 1 + |N|\}, \quad (25)$$

where $rand$ is a function returning a randomly generated number within $[0, 1]$. CR is the crossover rate which is distributed within $[0, 1]$ and mainly controls how many elements of the trial individual originates from the mutant individual. j_{rand} is an integer number randomly selected from $\{1, 2, 3, \dots, 1 + |N|\}$ to enforce that the trial individual differs from its corresponding parent individual in at least one element.

Selection Operator. Finally, to keep a constant population size in the following generations, the selection operator is executed to decide whether the trial or the parent individual survives to the next generation. In this paper, we employ an one-to-one selection operator. To be specific, each trial individual U_i^t is evaluated, and then its objective function value (reduced cost) $f(U_i^t)$ is compared with the objective function value of the corresponding parent individual $f(X_i^t)$ in the current population. If the objective function value of the trial individual is greater than or equal to that of the corresponding parent individual, then the parent individual is replaced by the trial one. Otherwise, the parent individual is preserved for the next generation. The selection operator can be described as follows.

$$X_{i,t+1} = \begin{cases} U_{i,t}, & \text{if } f(U_{i,t}) \geq f(X_{i,t}), \\ X_{i,t}, & \text{otherwise.} \end{cases} \quad (26)$$

Stopping Criteria of the DE Algorithm. In the proposed DE algorithm, two stopping conditions are adopted. If any of

these stopping conditions is met, the DE algorithm terminates. These two stopping conditions are as follows:

- (1) Maximum number of iterations: $Iter_{max}$ iterations
- (2) Maximum number of nonimprovement iterations: $Iter_{max-non}$ iterations

After termination of the DE algorithm, all columns with positive reduced cost that have been found in the final DE population are added to the RLMP.

4.5. Stopping Condition of the Column Generation Process. The column generation process which is an iterative procedure between the restricted master problem and sub-problems aims to identify columns with positive reduced cost. Therefore, we terminate the column generation process when it fails to find columns with positive reduced cost for a specified number of consecutive iterations.

4.6. Finding an Integer Solution. Column generation is used to solve the LP relaxation of the restricted master problem and thus may return a fractional solution. Therefore, when column generation reports a fractional solution, we adopt a primal heuristic to obtain a feasible integer solution based on the columns already enumerated in the RLMP. The heuristic simply imposes integrality to the master variables, converts the RLMP model to an IP model, and then calls an IP solver to solve the IP model. Since we are solving the IP model on a subset of columns, this process does not necessarily obtain

an optimal integer solution to the considered problem. However, it has been shown to be useful in practice, especially for the problem we address in this paper, as we will validate in Section 5.3. This heuristic has also been widely used in literature related to column generation-based heuristics (Hauge et al. [24]; Song et al. [25]; Liang et al. [26]).

5. Numerical Experiments

In this section, we conduct extensive computational experiments to verify the applicability and effectiveness of column generation based on randomly generated instances. Column generation is implemented in MATLAB 2016a and compared against the original MILP formulation solved by CPLEX 12.10. All runs are performed on a Windows 10 professional 64 bit operating system with an Intel Core i5-8250U CPU with 1.60 GHz and 8 GB RAM laptop. For each run, we enforce a time limit of 3600 s of CPU time.

5.1. Data Generation. There are no publicly available benchmark instances reflecting the setting of the studied problem. Therefore, we randomly generate instances in varying parameter and problem size via the generation schemes of Cesaret et al. [27] and Vallada et al. [19] with some adaptations to the characteristics of our problem. We generate three different sets: small-sized with $|N| = \{15, 20\}$ and $|M| = \{2, 3\}$, medium-sized with $|N| = \{40, 60\}$ and $|M| = \{2, 3\}$, and large-sized with $|N| = \{90, 100\}$ and $|M| = \{3, 4\}$.

Each $|N| \times |M|$ combination contains 9 subclasses, and each subclass comprises 5 randomly generated instances. Each subclass is characterized by parameters TF and R . More specifically, the parameter TF is the tardiness factor, which controls average due date values and parameter R determines the relative range of due dates. Parameters TF and R all take values from the set $\{0.2, 0.6, 1.0\}$, yielding the aforementioned 9 subclasses. Other parameters are generated as follows. Processing times are randomly generated from the interval $[1, 100]$, i.e., $p_{im} \sim U[1, 100]$, $i \in N$, $m \in M$. The due dates of jobs d_i are generated from the formula $d_i = r_i + \max\{\text{slack}, p_{im}\}$. slack is drawn from a discrete uniform distribution in the interval $[\bar{p}(1 - TF - R/2), \bar{p}(1 - TF + R/2)]$, where $\bar{p} = \sum_{i \in N} \sum_{m \in M} p_{im} / |M|^2$, similar to the study by Cesaret et al. [27] and Wang and Ye [9]. The release time r_i is generated from the interval $[0, TF \cdot \bar{p}]$. The number of resource types is set to be 2. For each type of resources, the resource consumption of each job is randomly generated in the interval $[1, 9]$. And, the total number of each type of resources is set to be $6 \times |M|$. Penalties and revenues are randomly generated from a discrete uniform distribution in the interval $[1, 10]$.

5.2. Parameter Settings and Performance Metrics

5.2.1. Parameter Settings. The parameter settings affect the quality of the solutions obtained by the proposed algorithm. In this paper, the Taguchi method is applied to tune the parameters. The Taguchi method which is essentially a

fractional factorial experiment approach uses orthogonal arrays to drastically reduce the number of experiments to be performed. The Taguchi method supposes two types of factors affect a process, i.e., controllable factors and uncontrollable noise factors. Since removal of the noise factors is impractical, the Taguchi method seeks to determine the optimal setting of controllable factors and to minimize the effect of noise, in order to achieve a robust performance. The responses at each setting of controllable parameters are considered as a measure, indicative of not only the mean of the performance but also its variance. The mean and the variance are transformed to a single performance measure known as the signal-to-noise (S/N) ratio. The term “signal” signifies the desirable value (mean response variable) and “noise” signifies the undesirable value (standard deviation), so the S/N ratio represents the amount of variation presents in the response variable. Thus, the goal is to maximize the S/N ratio.

This paper uses the Taguchi method as a reliable method of setting the parameters for the proposed algorithm. The DE algorithm has five parameters: (1) population size NP ; (2) scaling factor F ; (3) crossover rate CR ; (4) maximum number of iterations $Iter_{\max}$; and (5) maximum number of non-improvement iterations $Iter_{\max-\text{non}}$. And, the stopping condition of the column generation process involves one parameter $Iter_{\text{nonCG}}$, i.e., the number of consecutive iterations of failing to find a column with positive reduced cost. Five levels are considered for each parameter, as shown in Table 2. The orthogonal array $L_{25}(6^5)$ shown in Table 3 is designed as the experimental layout, which displays 25 different combinations of factor levels for the experiments.

For each of three different-sized problems, five instances are randomly selected and then tested on the 25 parameter combinations. The obtained computational data are processed by Minitab 19 statistical software. The S/N ratio trends of parameter levels for different-sized instances are displayed in Figures 4–6. As indicated in the three figures, we set $NP = 20$, $F = 0.6$, $CR = 0.75$, $Iter_{\max} = 100$, $Iter_{\max-\text{non}} = 50$, and $Iter_{\text{nonCG}} = 10$ for small-sized and medium-sized instances. And, for large-sized instances, we set $NP = 30$, $F = 0.6$, $CR = 0.75$, $Iter_{\max} = 500$, $Iter_{\max-\text{non}} = 100$, and $Iter_{\text{nonCG}} = 20$, respectively.

5.2.2. Performance Metrics. We use the gap between the revenue and the upper bound as the performance metric. The revenue is obtained from a given solution method, including either the proposed column generation approach or the integer programming formulation solved by CPLEX solver. The upper bound is obtained by solving the MILP formulation in Section 3 with a time limit of 3600 s. Furthermore, the number of optimal solutions out of 5 instances of each subclass and the average computation time are recorded as auxiliary metrics.

5.3. Computational Results and Analysis

5.3.1. Results of Small-Sized Instances. We first verify the performance of the proposed column generation approach

TABLE 2: Levels of parameters.

Parameter (factor)	NP	F	CR	$Iter_{\max}$	$Iter_{\max-\text{non}}$	$Iter_{\text{nonCG}}$	
Level	1	10	0.2	0.25	100	10	5
	2	15	0.4	0.50	200	25	10
	3	20	0.6	0.75	300	50	20
	4	25	0.8	0.95	400	75	30
	5	30	1	1	500	100	40

TABLE 3: The orthogonal array.

Experiment no.	Parameter					
	NP	F	CR	$Iter_{max}$	$Iter_{max-non}$	$Iter_{nonCG}$
1	1	1	1	1	1	1
2	1	2	2	2	2	2
3	1	3	3	3	3	3
4	1	4	4	4	4	4
5	1	5	5	5	5	5
6	2	1	2	3	4	5
7	2	2	3	4	5	1
8	2	3	4	5	1	2
9	2	4	5	1	2	3
10	2	5	1	2	3	4
11	3	1	3	5	2	4
12	3	2	4	1	3	5
13	3	3	5	2	4	1
14	3	4	1	3	5	2
15	3	5	2	4	1	3
16	4	1	4	2	5	3
17	4	2	5	3	1	4
18	4	3	1	4	2	5
19	4	4	2	5	3	1
20	4	5	3	1	4	2
21	5	1	5	4	3	2
22	5	2	1	5	4	3
23	5	3	2	1	5	4
24	5	4	3	2	1	5
25	5	5	4	3	2	1

FIGURE 4: The S/N ratio trend for small-sized problems.

FIGURE 5: The S/N ratio trend for medium-sized problems.

FIGURE 6: The S/N ratio trend for large-sized problems.

(CG for short) for solving small-sized instances. Tables 4 and 5 display the comparison results between the MILP formulation and the proposed column generation approach. The first column represents the $|N| \times |M|$ combinations. Column 2 and column 3 show parameter TF and parameter R , respectively. We report in columns 4–9 the minimum (Min), average (Avg) and maximum (Max) gaps between the upper bound and the best found solution obtained by the respective method for every subclass. In columns 10 and 11, we display the number of optimal solutions found by the given solution method. The following two columns with the same heading “Total” report the average computation time consumed by solution approaches. The remaining columns show the time spent on each step of column generation.

Specifically, columns “CH,” “LP,” “MILP,” and “DE” report the time spent by the constructive heuristic, LP solver, MILP solver, and the proposed DE algorithm, respectively.

As indicated by the gap values in Tables 4 and 5, both the MILP formulation and column generation solve all instances to optimality. In comparison with the MILP formulation, column generation consumes significantly less running time to find the optimal solutions. That is, the MILP formulation needs computing time increasing from few seconds to hundreds of seconds, whereas column generation is able to find the optimal solution in seconds. For example, the average CPU time reduces from 119.14 s with the MILP formulation to 3.24 s with the proposed algorithm for instances with $|N| = 20$, $|M| = 2$, $TF = 0.2$, and $R = 0.2$.

TABLE 4: Computation results of small-sized instances with $|N| = 15$.

$ N \times M $	TF	R	Gap (%)						No. of optimal solutions		CPU (s)					
			MILP			CG					MILP			CG		
			Min	Ave	Max	Min	Ave	Max	MILP	CG	Total	Total	CH	LP	MILP	DE
15×2	0.2	0.2	0	0	0	0	0	0	5	5	50.80	3.74	0.03	0.82	1.50	1.38
		0.6	0	0	0	0	0	0	5	5	78.68	3.46	0.03	1.33	0.51	1.59
		1	0	0	0	0	0	0	5	5	31.00	3.81	0.02	1.06	1.20	1.52
	0.6	0.2	0	0	0	0	0	0	5	5	74.30	2.20	0.03	0.67	0.25	1.25
		0.6	0	0	0	0	0	0	5	5	82.75	6.17	0.1	2.57	1.21	2.29
		1	0	0	0	0	0	0	5	5	21.00	6.42	0.02	3.50	0.45	2.45
	1	0.2	0	0	0	0	0	0	5	5	9.00	1.79	0.02	0.47	0.08	1.22
		0.6	0	0	0	0	0	0	5	5	67.52	2.94	0.02	1.09	0.05	1.78
		1	0	0	0	0	0	0	5	5	11.80	3.57	0.02	1.29	0.14	2.13
	15×3	0.2	0.2	0	0	0	0	0	0	5	5	15.21	3.25	0.03	1.09	0.59
0.6			0	0	0	0	0	0	5	5	9.53	2.04	0.03	0.55	0.23	1.23
1			0	0	0	0	0	0	5	5	10.86	1.79	0.03	0.49	0.16	1.11
0.6		0.2	0	0	0	0	0	0	5	5	12.32	1.39	0.02	0.31	0.08	0.97
		0.6	0	0	0	0	0	0	5	5	11.06	5.74	0.03	2.64	0.19	2.88
		1	0	0	0	0	0	0	5	5	18.62	1.84	0.02	0.48	0.19	1.14
1		0.2	0	0	0	0	0	0	5	5	13.68	1.34	0.02	0.24	0.08	1.00
1	0.6	0	0	0	0	0	0	5	5	10.22	1.75	0.02	0.38	0.10	1.25	
1	1	0	0	0	0	0	0	5	5	11.85	1.23	0.02	0.22	0.05	0.94	
Average	—	—	0	0	0	0	0	0	—	—	30.01	3.03	0.03	1.07	0.39	1.54

TABLE 5: Computation results of small-sized instances with $|N| = 20$.

$ N \times M $	TF	R	Gap (%)						No. of optimal solutions		CPU(s)					
			MILP			CG			MILP	CG	MILP			CG		
			Min	Ave	Max	Min	Ave	Max			Total	Total	CH	LP	MILP	DE
20×2	0.2	0.2	0	0	0	0	0	0	5	5	119.14	3.24	0.03	1.08	0.68	1.45
		0.6	0	0	0	0	0	0	5	5	68.63	4.49	0.03	1.91	0.64	1.91
		1	0	0	0	0	0	0	5	5	32.27	3.58	0.03	1.30	0.28	1.97
	0.6	0.2	0	0	0	0	0	0	5	5	45.83	6.11	0.04	1.90	2.25	1.92
		0.6	0	0	0	0	0	0	5	5	22.23	4.88	0.02	2.46	0.77	1.63
		1	0	0	0	0	0	0	5	5	42.65	5.05	0.02	2.04	0.46	2.53
	1	0.2	0	0	0	0	0	0	5	5	20.60	6.55	0.02	2.46	0.05	4.02
		0.6	0	0	0	0	0	0	5	5	19.86	7.19	0.02	2.74	0.42	4.01
		1	0	0	0	0	0	0	5	5	29.80	5.83	0.02	2.23	0.32	3.26
	20×3	0.2	0.2	0	0	0	0	0	0	5	5	45.21	5.24	0.07	1.53	0.54
0.6			0	0	0	0	0	0	5	5	39.03	5.28	0.12	1.74	0.32	3.10
1			0	0	0	0	0	0	5	5	71.07	6.27	0.03	1.95	0.61	3.68
0.6		0.2	0	0	0	0	0	0	5	5	48.03	6.03	0.03	1.95	0.78	3.27
		0.6	0	0	0	0	0	0	5	5	57.81	6.75	0.02	2.05	0.51	4.17
		1	0	0	0	0	0	0	5	5	45.02	5.71	0.02	1.58	0.57	3.54
1		0.2	0	0	0	0	0	0	5	5	36.20	6.82	0.02	2.10	0.15	4.55
		0.6	0	0	0	0	0	0	5	5	37.69	3.19	0.02	0.68	0.11	2.38
		1	0	0	0	0	0	0	5	5	42.03	3.32	0.13	0.61	0.18	2.40
Average	—	—	0	0	0	0	0	—	—	45.73	5.31	0.04	1.80	0.54	2.94	

To sum up, the numerical experiments on small-sized instances show that the proposed algorithm significantly outperforms the MILP formulation in terms of computational efficiency.

5.3.2. Results of Medium-Sized Instances. We further examine the performances of the proposed method and the

MILP formulation for solving medium-sized instances. Tables 6 and 7 present the experiment results.

As shown in Tables 6 and 7, we can observe that the proposed column generation algorithm is superior to the MILP formulation in terms of efficiency and effectiveness. For instances with $|N| = 40$, column generation finds 83 optimal solutions out of 90 instances with 0.13% deviation on average. In contrast, the MILP formulation identifies the

TABLE 6: Computation results of medium-sized instances with $|N| = 40$.

$ N \times M $	TF	R	Gap (%)						No. of optimal solutions		CPU(s)					
			MILP			CG					MILP			CG		
			Min	Ave	Max	Min	Ave	Max	$MILP$	CG	$Total$	$Total$	CH	LP	$MILP$	DE
40×2	0.2	0.2	0.00	2.13	10.66	0	0	0	4	5	1610.20	30.18	0.11	3.21	25.07	1.79
		0.6	0	0	0	0	0	0	5	5	134.47	10.18	0.07	3.55	4.42	2.14
		1	0	0	0	0	0	0	5	5	213.20	10.25	0.12	3.48	4.02	2.63
	0.6	0.2	0	14.68	33.19	0	1.33	3.91	2	2	2204.82	358.08	0.14	2.96	347.32	7.66
		0.6	0	3.79	17.45	0	0.35	0.88	3	3	447.67	87.96	0.07	2.59	18.79	66.52
		1	0	10.37	22.02	0	0.66	1.96	1	3	2907.40	52.63	0.07	4.18	16.70	31.68
	1	0.2	0	0	0	0	0	0	5	5	758.61	23.04	0.03	11.89	0.44	10.68
		0.6	0	0	0	0	0	0	5	5	182.65	16.58	0.03	5.29	1.13	10.13
		1	0	0	0	0	0	0	5	5	110.00	14.02	0.20	2.94	0.78	10.10
	40×3	0.2	0.2	0	0	0	0	0	0	5	5	96.61	9.87	0.08	4.09	2.91
0.6			0	0	0	0	0	0	5	5	92.25	5.58	0.09	2.67	0.68	2.14
1			0	0	0	0	0	0	5	5	130.82	2.38	0.07	0.73	0.18	1.40
0.6		0.2	0	0	0	0	0	0	5	5	107.20	10.61	0.05	4.30	3.32	2.94
		0.6	0	0	0	0	0	0	5	5	239.68	8.02	0.04	3.18	1.96	2.84
		1	0	0	0	0	0	0	5	5	75.25	14.00	0.05	7.95	1.78	4.22
		0.2	0	0	0	0	0	0	5	5	77.21	7.16	0.04	2.99	0.32	3.81
1	0.6	0	0	0	0	0	0	5	5	96.63	52.56	0.04	6.00	1.25	45.27	
	1	0	0	0	0	0	0	5	5	92.20	3.46	0.03	1.28	0.30	1.85	
Average	—	—	0	1.72	4.63	0	0.13	0.38	—	—	532.05	39.81	0.07	4.07	23.97	11.70

TABLE 7: Computation results of medium-sized instances with $|N| = 60$.

$ N \times M $	TF	R	Gap (%)						No. of optimal solutions		CPU(s)						
			MILP			CG					MILP			CG			
			Min	Ave	Max	Min	Ave	Max	$MILP$	CG	$Total$	$Total$	CH	LP	$MILP$	DE	
60×2	0.2	0.2	4.67	12.95	18.48	0	0	0	0	5	3600	103.26	0.14	6.28	93.93	2.91	
		0.6	6.32	13.39	20.34	0	0	0	0	5	3600	121.10	0.25	6.70	110.34	3.81	
		1	10.33	12.06	13.15	0	0	0	0	5	3600	80.23	0.27	5.95	70.30	3.71	
	0.6	0.2	15.68	18.95	24.54	0	0.88	2.11	0	2	3600	180.37	0.12	4.95	83.44	91.86	
		0.6	12.57	15.97	20.85	0	0.41	2.07	0	4	3600	144.27	0.08	4.41	125.42	14.36	
		1	9.86	14.31	17.82	0	0	0	0	5	3600	20.02	0.12	9.54	5.09	5.27	
	1	0.2	0	0	0	0	0	0	5	5	417.89	107.97	0.05	69.93	6.67	31.32	
		0.6	0	21.28	31.61	0	1.03	2.50	1	1	3600	59.8	0.05	12.19	3.91	43.65	
		1	7.03	16.55	25.46	0	0.52	1.32	0	3	3600	23.33	0.08	10.40	5.51	7.34	
	0.2	0	0	0	0	0	0	5	5	971.81	11.61	0.17	5.53	2.31	3.60		
60×3	0.2	0.6	0	5.64	8.69	0	0	0	1	5	2961.22	12.63	0.17	6.70	1.87	3.89	
		1	0	6.62	17.39	0	0	0	2	5	2287.86	10.53	0.11	4.77	2.03	3.62	
		0.2	0	0	0	0	0	0	5	5	221.21	28.93	0.05	5.68	19.88	3.32	
	0.6	0.6	0	13.71	26.03	0	0	0	1	5	2926.07	10.19	0.08	5.14	1.73	3.24	
		1	0	14.28	33.12	0	0	0	2	5	2328.03	12.19	0.05	6.46	1.79	3.89	
		0.2	0	0	0	0	0	0	5	5	228.09	18.75	0.02	9.63	0.69	8.41	
	1	0.6	0	0	0	0	0	0	5	5	235.21	10.96	0.03	5.46	1.24	4.23	
		1	0	2.75	13.77	0	0.26	1.33	4	4	935.25	8.39	0.04	3.83	0.80	3.72	
		Average	—	—	3.69	9.36	15.07	0	0.17	0.52	—	—	2350.70	53.59	0.10	10.20	29.83

optimal solution in 80 out of 90 instances and reports 1.72% deviation on average. As expected, the performance of the MILP formulation degenerates since the solution space exponentially grows with the increase in problem size. When $|N|$ equals 60, the MILP formulation only reports 36 instances out of 90 to optimality within 3600 s, whereas column generation finds 79 optimal solutions. The average CPU

time of the MILP formulation increases to 2350.70 s, while column generation runs in 53.59 s on an average.

Based on above comparison analysis, we can come to the conclusion that column generation still performs significantly better than the MILP formulation with regards to efficiency and effectiveness and is an effective approach for solving medium-sized instances.

TABLE 8: Computation results of large-sized instances with $|N| = 90$.

$ N \times M $	TF	R	Gap (%)						No. of optimal solutions		CPU(s)					
			MILP			CG					MILP			CG		
			Min	Ave	Max	Min	Ave	Max	$MILP$	CG	$Total$	$Total$	CH	LP	$MILP$	DE
90×3	0.2	0.2*	1.60	8.56	11.68	0	0	0	0	5	2082.23	357.67	0.21	301.68	12.66	43.12
		0.6*	2.94	5.18	6.30	0	0	0	0	5	1148.00	296.32	0.18	239.84	10.71	45.59
		1 *	0.39	5.83	11.97	0	0	0	0	5	2100.61	218.42	0.19	160.60	7.19	50.44
		0.2	0	11.74	18.28	0	0	0	1	5	2880.21	248.82	0.07	192.12	19.90	36.73
		0.6	8.33	12.87	16.63	0	0	0	0	5	3600	223.36	0.06	177.00	9.51	36.79
		1 *	7.73	14.48	20.38	0	0	0	0	5	2964.35	228.28	0.07	173.02	7.57	47.62
	1	0.2	0	0.11	0.55	0	0.06	0.30	4	4	1080.89	121.08	0.04	63.77	10.48	46.79
		0.6	0	6.69	19.25	0	0	0	3	5	1848.10	107.61	0.05	62.12	3.71	41.73
		1 *	11.19	13.13	15.63	0	0	0	0	5	2412.80	133.22	0.04	77.58	9.30	46.30
		0.2	0	5.82	14.11	0	0	0	0	5	3600	316.04	0.13	254.71	7.06	54.14
		0.6	1.30	3.68	11.07	0	0	0	0	5	3600	274.33	0.14	214.28	7.76	52.15
		1*	-	-	-	0	0	0	0	5	414.81	225.31	0.12	164.48	4.54	56.17
90×4	0.6	0.2	0	6.76	15.99	0	0	0	2	5	2376.00	308.20	0.05	243.17	9.73	55.25
		0.6	0	5.00	13.70	0	0	0	1	5	1762.24	271.05	0.04	200.51	9.40	61.10
		1	0	17.51	78.60	0	0	0	1	5	1139.31	189.58	0.04	125.93	5.68	57.93
	1	0.2	0	0	0	0	0	0	5	5	302.12	67.80	0.04	29.61	1.53	36.62
		0.6	0	0	0	0	0	0	5	5	408.08	88.78	0.03	43.62	3.08	42.05
		1	0	4.49	11.29	0	0	0	2	5	2423.28	112.08	0.04	58.96	4.43	48.65
Average	—	—	1.97	7.17	15.61	0	0.00	0.02	—	—	2007.95	210.44	0.09	154.61	8.01	47.73

“-” indicates that the CPLEX fails to find feasible solutions for this subclass. “*” indicates that the CPLEX runs out of memory when solving some instances of this subclass.

TABLE 9: Computation results of large-sized instances with $|N| = 100$.

$ N \times M $	TF	R	Gap (%)						No. of optimal solutions	CPU(s)							
			MILP			CG				MILP			CG				
			Min	Ave	Max	Min	Ave	Max		$MILP$	CG	$Total$	$Total$	CH	LP	$MILP$	DE
100×3	0.2	0.2*	2.24	8.00	13.92	0	0	0	0	5	571.84	374.27	0.25	208.58	10.50	154.94	
		0.6*	5.38	7.31	8.67	0	0	0	0	5	627.31	247.86	0.24	147.80	10.56	89.26	
		1*	-	-	-	0	0	0	0	5	436.57	200.05	0.26	129.56	8.90	61.33	
		0.2	10.04	15.24	19.45	0	0	0	0	5	3600	359.94	0.1	192.67	122.71	44.46	
		0.6	10.37	15.57	20.41	0	0	0	0	5	3600	254.40	0.07	185.37	12.27	56.69	
		1*	14.24	15.05	16.00	0	0	0	0	5	1711.34	248.25	0.07	165.48	11.87	70.83	
	1	0.2*	0	0.41	1.35	0	0.32	0.86	3	3	1198.63	147.54	0.06	73.95	12.42	61.11	
		0.6*	0	8.54	18.24	0	0	0	1	5	2454.21	160.96	0.09	88.21	12.60	60.06	
		1*	10.02	10.81	12.25	0	0	0	0	5	1055.10	176.53	0.05	99.08	7.47	69.93	
		0.2*	3.97	5.57	7.04	0	0	0	0	5	1751.80	488.65	0.32	389.70	12.00	86.63	
		0.2	0.6*	0.82	4.79	8.75	0	0	0	0	5	519.20	489.11	0.25	376.24	14.41	98.21
		1*	-	-	-	0	0	0	0	5	457.40	402.91	0.35	264.66	12.91	124.99	
100×4	0.6	0.2	0	7.49	13.13	0	0	0	1	5	3000	793.43	0.06	619.86	24.58	148.93	
		0.6*	9.51	12.12	17.90	0	0	0	0	5	2977.87	562.54	0.09	395.90	20.20	146.35	
		1*	7.54	7.54	7.54	0	0	0	0	5	438.96	337.99	0.08	142.00	21.61	174.30	
	1	0.2	0	0	0	0	0	0	5	5	410.21	121.41	0.06	39.76	6.27	75.32	
		0.6	0	1.28	6.42	0	0	0	4	5	1164.28	141.24	0.06	73.00	6.22	61.96	
		1*	0.55	5.62	9.39	0	0	0	0	5	1073.99	155.21	0.07	78.84	6.34	69.96	
Average	—	—	4.67	7.83	11.28	0	0.02	0.05	—	—	1502.71	314.57	0.14	203.93	18.55	91.96	

“-” indicates that the CPLEX fails to find feasible solutions for this subclass. “*” indicates that the CPLEX runs out of memory when solving some instances of this subclass.

5.3.3. Results of Large-Sized Instances. We further report computational results obtained for large-sized instances in Tables 8 and 9. These instances become more challenging for the MILP formulation, which can be indicated by computation time. We can observe that column generation performs significantly better than the MILP formulation in

terms of number of instances solved to optimality and computation times. Specifically, the average percentage deviation of the MILP formulation is consistently larger than that of column generation. We can also observe that the MILP formulation fails to find a feasible solution to instances with $TF = 0.2$, $R = 1$ and $|N| = 100$. In contrast, the proposed

column generation algorithm is able to prove the optimality of 177 instances and to provide very low gaps for the remaining, always below 1%. Column generation can generally solve each instance within 800 s, which further demonstrates its efficiency.

The encouraging computational results of the proposed algorithm may be attributed to two aspects. Firstly, the column generation approach keeps most columns of the set partitioning formulation implicit and only generates necessary columns that can improve the objective value. Therefore, a considerable number of times can be saved. Secondly, the DE algorithm is an efficient heuristic. It is used to solve the subproblems and helps to identify promising columns in a very short time.

In summary, the capability of the proposed algorithm to consistently find optimal or near-optimal solutions confirms its effectiveness and superiority against the MILP formulation. Our experimental results clearly show that column generation is a computationally viable solution approach for practical situations.

5.4. Effect of Parameters TF and R on the Performance of the Methods. In this section, we further investigate the impact of the instance generation parameters on the performance of the methods. As stated in Section 5.1, parameter TF determines due date tightness while parameter R controls the due date range. Additionally, parameter TF also controls the dispersion degree of the release dates of the jobs, in a way that the higher the TF is, the more dispersed the release dates are.

As indicated by the gap and computation time values in Tables 4–9, we can notice that given that all other specifications are the same, instances with large values of TF are generally easier to solve than those with small values. A reasonable explanation for this behaviour could be as follows. A small TF value corresponds to a situation where the release dates are distributed in the very beginning of the time horizon and then are not very restrictive. And, a very large TF value leads to a situation where the release dates are very dispersed. Hence, as TF increases, we receive incoming orders released at dispersed time points with tight due dates. Therefore, more orders are expected to be rejected, which in turn implies the reduction of the size of the solution space. Consequently, the instances with large TF values are less challenging. We can observe from the tables that the number of instances solved to optimality by the MILP formulation increases and the average CPU time spent by the MILP formulation tends to decrease as the TF gets large when it comes to medium- and large-sized instances. A similar observation can be found when instances are solved by the proposed column generation algorithm. That is, two subclasses are all exactly solved by the proposed algorithm, but the subclass with $TF = 1$ evidently consumes less CPU time than the subclass with $TF = 0.2$.

Given the TF value, the due date range increases with the increase of the parameter R . Results in Tables 4–9 show that given the TF value, some subclasses with large R value are more difficult to solve than those with small R value, but we

can also find some exceptions. This coincides with observations reported by Wang and Ye [9]. Therefore, there is no clear trend in correlation between the R value and the hardness of the problem.

6. Conclusion

In this paper, we study a resource-constrained order acceptance and scheduling on unrelated parallel machines. We present two different models for this problem, i.e., the MILP formulation and the set partitioning formulation. With the MILP formulation including all decision variables, we can directly solve problem instances via general-purpose commercial solvers without complicated programming. In order to solve large-scale instances, we develop a column generation algorithm based on the set partitioning formulation. In the proposed column generation algorithm, we propose a constructive heuristic to initialize the restricted linear master problem and design a differential evolution algorithm to solve the subproblems efficiently. Numerical experiments based on a set of randomly generated test instances are carried out to verify the performance of the proposed algorithm. The results demonstrate that the MILP formulation can identify optimal solutions for small-scale instances and report solutions with large optimality gaps at termination for medium- and large-sized instances. Column generation is advantageous over the MILP formulation in terms of effectiveness and efficiency and quite suitable for problems with different sizes. We also analyse the effect of parameters TF and R on the performance of the proposed algorithm and the MILP formulation.

Further research will concentrate on problem extensions and algorithmic enhancements. One possible extension of the current work is to consider stochasticities in processing times and develop a stochastic and robust counterpart for the column generation algorithm. Considering some practical assumptions including machine breakdown, parallel batch processing, and order interruption are complex but extremely valuable variations to the studied problem. Furthermore, we also attempt to improve the performance of the proposed column generation algorithm by solving the subproblems via a dynamic algorithm in conjunction with heuristics.

Data Availability

All data used to support the findings of the study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (no. 72101152) and by a grant from the research fund for Young Scholars in Shanghai (no. ZZLX20013).

References

- [1] S. A. Slotnick, "Order acceptance and scheduling: a taxonomy and review," *European Journal of Operational Research*, vol. 212, no. 1, pp. 1–11, 2011.
- [2] L. Fanjul-Peyro and R. Ruiz, "Scheduling unrelated parallel machines with optional machines and jobs selection," *Computers & Operations Research*, vol. 39, no. 7, pp. 1745–1753, 2012.
- [3] C. J. Hsu and C. W. Chang, "Unrelated parallel-machine scheduling with deteriorating jobs and rejection," *Applied Mechanics and Materials*, vol. 263, pp. 655–659, 2013.
- [4] D. Jiang and J. Tan, "Scheduling with job rejection and nonsimultaneous machine available time on unrelated parallel machines," *Theoretical Computer Science*, vol. 616, pp. 94–99, 2016.
- [5] S. Emami, M. Sabbagh, and G. Moslehi, "A Lagrangian relaxation algorithm for order acceptance and scheduling problem: a globalised robust optimisation approach," *International Journal of Computer Integrated Manufacturing*, vol. 29, no. 5, pp. 535–560, 2016.
- [6] S. Emami, G. Moslehi, and M. Sabbagh, "A Benders decomposition approach for order acceptance and scheduling problem: a robust optimization approach," *Computational and Applied Mathematics*, vol. 36, no. 4, pp. 1471–1515, 2017.
- [7] B. Wang and H. Wang, "Multiobjective order acceptance and scheduling on unrelated parallel machines with machine eligibility constraints," *Mathematical Problems in Engineering*, vol. 2018, Article ID 6024631, 12 pages, 2018.
- [8] B. Mor and G. Mosheiov, "A note: minimizing total absolute deviation of job completion times on unrelated machines with general position-dependent processing times and job-rejection," *Annals of Operations Research*, vol. 271, no. 2, pp. 1079–1085, 2018.
- [9] S. Wang and B. Ye, "Exact methods for order acceptance and scheduling on unrelated parallel machines," *Computers & Operations Research*, vol. 104, pp. 159–173, 2019.
- [10] J. A. Ventura and D. Kim, "Parallel machine scheduling about an unrestricted due date and additional resource constraints," *IIE Transactions*, vol. 32, no. 2, pp. 147–153, 2000.
- [11] J. A. Ventura and D. Kim, "Parallel machine scheduling with earliness-tardiness penalties and additional resource constraints," *Computers & Operations Research*, vol. 30, no. 13, pp. 1945–1958, 2003.
- [12] J.-F. Chen, "Unrelated parallel machine scheduling with secondary resource constraints," *International Journal of Advanced Manufacturing Technology*, vol. 26, no. 3, pp. 285–292, 2005.
- [13] J.-F. Chen and T.-H. Wu, "Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints," *Omega*, vol. 34, no. 1, pp. 81–89, 2006.
- [14] A. J. Ruiz-Torres, F. J. López, J. C. Ho, and J. C. Ho, "Scheduling uniform parallel machines subject to a secondary resource to minimize the number of tardy jobs," *European Journal of Operational Research*, vol. 179, no. 2, pp. 302–315, 2007.
- [15] E. B. Edis, C. Oguz, I. Ozkarahan, and I. Ozkarahan, "Parallel machine scheduling with additional resources: notation, classification, models and solution methods," *European Journal of Operational Research*, vol. 230, no. 3, pp. 449–463, 2013.
- [16] M. Afzalirad and J. Rezaeian, "Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions," *Computers & Industrial Engineering*, vol. 98, pp. 40–52, 2016.
- [17] J. Li, Y. Huang, and X. Niu, "A branch population genetic algorithm for dual-resource constrained job shop scheduling problem," *Computers & Industrial Engineering*, vol. 102, pp. 113–131, 2016.
- [18] L. Fanjul-Peyro, F. Perea, and R. Ruiz, "Models and mathematical heuristics for the unrelated parallel machine scheduling problem with additional resources," *European Journal of Operational Research*, vol. 260, no. 2, pp. 482–493, 2017.
- [19] E. Vallada, F. Villa, and L. Fanjul-Peyro, "Enriched meta-heuristics for the resource constrained unrelated parallel machine scheduling problem," *Computers & Operations Research*, vol. 111, pp. 415–424, 2019.
- [20] S. Nguyen, D. Thiruvady, A. T. Ernst, and D. Alahakoon, "A hybrid differential evolution algorithm with column generation for resource constrained job scheduling," *Computers & Operations Research*, vol. 109, pp. 273–287, 2019.
- [21] A. Yepes-Borrero, C. Juan, F. Villa, F. Perea, and J. P. Caballero-Villalobos, "GRASP algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources," *Expert Systems with Applications*, vol. 141, pp. 1–12, 2020.
- [22] C.-F. Liaw, Y.-K. Lin, C.-Y. Cheng, and M. Chen, "Scheduling unrelated parallel machines to minimize total weighted tardiness," *Computers & Operations Research*, vol. 30, no. 12, pp. 1777–1789, 2003.
- [23] R. Storn and K. Price, "Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.
- [24] K. Hauge, J. Larsen, R. M. Lusby, and E. Krappner, "A hybrid column generation approach for an industrial waste collection routing problem," *Computers & Industrial Engineering*, vol. 71, pp. 10–20, 2014.
- [25] Y. Song, J. Zhang, Z. Liang, and C. Ye, "An exact algorithm for the container drayage problem under a separation mode," *Transportation Research Part E: Logistics and Transportation Review*, vol. 106, pp. 231–254, 2017.
- [26] Z. Liang, F. Xiao, X. Qian et al., "A column generation-based heuristic for aircraft recovery problem with airport capacity constraints and maintenance flexibility," *Transportation Research Part B: Methodological*, vol. 113, pp. 70–90, 2018.
- [27] B. Cesaret, C. Oğuz, and F. Sibel Salman, "A tabu search algorithm for order acceptance and scheduling," *Computers & Operations Research*, vol. 39, no. 6, pp. 1197–1205, 2012.