Science Press

Contents lists available at ScienceDirect

Journal of Management Science and Engineering

journal homepage: www.keaipublishing.com/en/journals/journal-of-management-science-and-engineering/

A branch-and-price algorithm to perform single-machine scheduling for additive manufacturing

Lindong Liu, Zhenyu Wu, Yugang Yu[*]

School of Management, University of Science and Technology of China, Hefei, 230026, China

ARTICLE INFO

Article history:
Received 4 January 2022
Received in revised form 1 October 2022
Accepted 7 October 2022
Available online 16 December 2022

Keywords:
Single-machine scheduling
Branch and price
Labelling algorithm
Additive manufacturing

ABSTRACT

Additive manufacturing (AM) has attracted significant attention in recent years based on its wide range of applications and growing demand. AM offers the advantages of production flexibility and design freedom. In this study, we considered a practical variant of the batch-processing-machine (BPM) scheduling problem that arises in AM industries, where an AM machine can process multiple parts simultaneously, as long as the two-dimensional rectangular packing constraint is not violated. Based on the set-partitioning formulation of our mixed-integer programming (MIP) model, a branch-and-price (B&P) algorithm was developed by embedding a column-generation technique into a branch-and-bound framework. Additionally, a novel labelling algorithm was developed to accelerate the column-generation process. Ours is the first study to provide a B&P algorithm to solve the BPM scheduling problem in the AM industry. We tested the performance of our algorithm using a modern MIP solver (Gurobi) and real data from a 3D printing factory. The results demonstrate that for most instances tested, our algorithm produces results similar or identical to those of Gurobi with reasonable computation time and outperforms Gurobi in terms of solution quality and running time on some large instances.

1. Introduction

Additive manufacturing (AM), which is also known as 3D printing, differs from conventional manufacturing technologies from three perspectives: design freedom, quality differentiation, and natural flexibility (Dong et al., 2020). AM is a method of building parts in layer-by-layer fashion based on 3D digital models (Song and Zhang, 2020). This technique has gained widespread popularity for rapid prototyping. In recent years, AM has attracted significant attention for real production needs such as in the medical device, aerospace, and automobile industries. Furthermore, according to Larrañeta et al. (2020), AM can be used to mitigate the influence of potential pandemics and support supply chains, particularly during the COVID-19 crisis.

Modes of 3D printing include fused deposition modelling and stereolithography (SLA). In this study, we focused on the SLA AM process, which uses light to solidify liquid resin. Motivated by the efforts of our industry partner for production planning and machine scheduling in daily operations, we propose an exact algorithmic framework to tackle the AM machine-scheduling problem in a make-to-order environment. This problem belongs to the larger scope of batch-processing-machine (BPM) scheduling problems, where a machine can process multiple jobs simultaneously. A special BPM problem

[*] Corresponding author.
E-mail address: ygyu@ustc.edu.cn (Y. Yu).

https://doi.org/10.1016/j.jmse.2022.10.001

with two-dimensional rectangular packing is addressed, whereas most BPM problems are modelled as one-dimensional knapsack problems. BPM scheduling problems can be summarized as the assignment of a set of jobs to one or more machines and the sequencing of those jobs to optimize certain criteria such as earliness-tardiness and makespan (Muter, 2020). The BPM scheduling problem and 2D packing problem have been proven to be NP-hard and our problem remains the same.

In our setting, there is an AM machine and the input is a set of parts defined by standard triangle language (STL) files from customers. To focus on algorithm design, we simplify STL files as the minimum rectangular bounding boxes enclosing the parts. The production area of each part is represented by the projected area of the bounding box onto the machine-building platform and the packing of a group of parts is considered to be a 2D rectangular packing problem. The capacity of the machine is restricted by the two-dimensional area of the building platform and its height. The maximum height of the parts cannot exceed the height of the machine platform.

Scheduling decisions must be made to group different parts together to form a batch and allocate the batch to a printing machine. Parts from the same batch are placed on a platform without overlapping. The goal is to minimize the makespan for printing all parts. One interesting feature is that the processing time of a batch in the SLA process is not determined by the greatest processing time among the parts, but is related to the total volume of the parts, as well as the maximum height of those parts (Kucukkoc, 2019). The production speed considered in the SLA process is mainly related to the scanning speed (Sc) and recoating speed (Re). Because SLA builds parts in layer-by-layer fashion using liquid resin, the scanning time is the time required for laser application over the surface of the liquid resin, which is proportional to the total volume of the batch. The recoating time is the time required for the recoater to lower the machine platform, which is proportional to the maximum height of the parts in a batch, so a newly formed layer of liquid resin can be scanned.

The main contributions of this study are twofold. First, while most studies addressing AM machine-scheduling problems have focused on model construction and heuristic algorithms, our study uses an exact algorithm (branch-and-price (B&P)) framework to solve a new practical variant of the classical BPM scheduling problem in AM industries. Practical factors related to two-dimensional packing and processing time calculations are considered. Second, real data from our industry partner were used to validate the effectiveness of our algorithm.

The remainder of this paper is organized as follows. Section 2 reviews related works. A formal description and mathematical formulation are presented in Section 3, followed by a description of the B&P algorithm in Section 4. Section 5 presents numerical experiments conducted to evaluate the performance of the proposed algorithm. Finally, our conclusions are summarized in Section 6.

2. Related works

Over the past few decades, BPM scheduling has been explored extensively. The single-BPM scheduling problem can be denoted as $1|s_j, B|C_{\max}$ and was first proposed by Uzsoy to minimize the makespan of burn-in operations in semiconductor manufacturing (Uzsoy, 1994). Uzsoy proved that $C_{\max}$ is strongly NP-hard and proposed a heuristic solution. Since then, many research efforts have been devoted to heuristics and meta-heuristics for BPM scheduling problems, including genetic algorithms (Wang and Uzsoy, 2002; Kashan et al., 2006; Damodaran et al., 2006), tabu search (Meng and Tang, 2010), ant colony algorithms (Xu et al., 2012), particle swarm optimization (Beldar and Costa, 2018), and simulated annealing (Jin et al., 2009; Ghodratnama et al., 2010). Recently, intelligent algorithms were used by Rafiee Parsa et al. (2021) to solve a single-BPM scheduling problem and a neural-network-based approach was proposed. Trindade et al. (2021) used an arc-flow-based optimization approach and constructed a novel formulation of the problem as determining flows in graphs. They proved their approach's optimality for random instances with up to 100 million jobs.

Existing methods for BPM scheduling differ in many aspects, including their decision criteria, machine type, processing time, and job release time. The problem of scheduling a single BPM with incompatible job families was studied by Uzsoy (1995), Azizoglu and Webster (2001), and Liu (2013), who determined that jobs of different families cannot be processed together in the same batch. Dobson and Nambimadom (2001) discussed the problem of batching and scheduling, where each job is associated with a given family and has a weight or delay cost. Zarook et al. (2015) considered the machine aging effect and multi-maintenance activity concept. Some researchers have also focused on online scheduling by considering job release time (Ozturk et al., 2012; Li et al., 2005; Beldar and Costa, 2018). Wang and Uzsoy (2002) developed a genetic algorithm to address the problem of minimizing the maximum lateness of a batch processing machine with dynamic job arrivals. Zheng et al. (2021) investigated a single-batch scheduling problem with dual quantitative and technological setup times extracted from autoclave molding production.

While many studies have applied heuristic and meta-heuristic algorithms to tackle the NP-hard problem $1|s_j,B|C_{\max}$, some exact methods have also been developed. Both dynamic programming and branch-and-bound algorithms were proposed by Rafiee Parsa et al. (2017) to solve a just-in-time BPM scheduling problem with the objective of minimizing the total earliness and tardiness of jobs. Azizoglu and Webster (2001) proposed a branch-and-bound procedure applicable to a batch processor model to minimize the total weighted completion time. Tanaka et al. (2009) proposed several improvements to the original successive sublimation dynamic programming method to reduce both memory usage and computational load. Their algorithm can optimally solve 300 jobs instances of the total weighted earliness-tardiness problem. Rafiee Parsa et al. (2010) proposed a B&P algorithm for the $1|s_j, B|C_{\max}$ problem based on a combination of the column-generation technique with the branch-and-bound method to obtain an optimal solution for the original mixed-integer programming (MIP) model following set partitioning formulation. The column-generation-based approaches mentioned above have been extensively applied to solve complex combinatorial optimization problems such as the cutting stock problem (Degraeve and Peeters, 2003), machine-scheduling problems (Chen and Powell, 1999), and vehicle routing problems (Faiz et al., 2019). However, thus far, no exact methods have been developed to address the BPM scheduling problem with two-dimensional packing.

Parallel BPM scheduling problems have also attracted the attention of researchers. We refer interested readers to the paper by Fowler and Lars Mönch (2022) for an in-depth review of this subject. Fanjul-Peyro et al. (2019) proposed novel mixed-integer linear programs and a mathematical-programming-based algorithm for unrelated parallel machine-scheduling problems, which can obtain near-optimal solutions for up to 1000 jobs on eight machines with relative deviations from the lower bounds below 0.8%. Zhang et al. (2020) considered machine scheduling for AM, but this problem is difficult to solve based on the requirement for simultaneous multi-part assignment and placement, so the authors proposed an improved evolutionary algorithm by combining a genetic algorithm with a heuristic placement strategy. More recently, Che et al. (2021) developed a simulated annealing method with predefined packing strategies to solve a problem similar to the orientation selection of 3D printed parts.

The main contributions of this study can be summarized as follows. First, a B&P algorithm is proposed to tackle a difficult 3D printing machine-scheduling problem. Practical factors related to two-dimensional rectangular packing and processing time calculation are included in our model, which accurately reflects the reality of the 3D printing process, a much simpler version of which was proven to be NP-Hard (Uzsoy, 1994). Second, extensive numerical experiments are presented based on a real industry dataset to demonstrate the performance of the proposed algorithm.

3. Problem description

The target problem can be denoted as a single-BPM scheduling problem in which the objective is to minimize the makespan ($C_{\max}$). Each part $i \in N$ has a fixed length, width, and height, which are denoted as l_i, w_i, and h_i, respectively, with a specific orientation. The volume of part i is v_i. A BPM can process a group of parts simultaneously, given that the parts allocated to the same batch do not have any overlap. The processing time of batch k is related to the total volume (V_k) of the parts, as well as the maximum height (H_k) of the parts allocated to that batch. The binary decision variable α_{ik} is defined as

$$\alpha_{ik} = \begin{cases} 1 \textit{ if part i is assigned to batch k} \\ \quad\quad\quad 0 \textit{ otherwise} \end{cases}$$

The variables and parameters used in our mathematical model are defined as follows:

- β_k: binary variable equal to one if batch k is used and equal to zero otherwise;
- PL_{ij}: binary variable equal to one if part i is on the left of part j and equal to zero otherwise;
- PB_{ij}: binary variable equal to one if part i is below part j and equal to zero otherwise;
- V_k, H_k: total volume and maximum height of the k_{th} batch of the machine;
- (x_i, y_i): coordinates of the bottom-left corner of part i;
- ψ: a sufficiently large positive number.

The MIP model is defined as follows:

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

$$\text{s.t.} C_{max} \geq C_k, \forall k \in K \tag{2}$$

$$C_1 = p_1, \tag{3}$$

$$C_k = C_{k-1} + p_k, \forall k \in K \tag{4}$$

$$p_k = t + Re \cdot max_{i \in I}\{h_i \alpha_{ik}\} + Sc \cdot \sum_{i=1}^{n} v_i \alpha_{ik}, \forall i \in I, k \in K. \tag{5}$$

$$H_k \geq h_i - \Psi(1 - \alpha_{ik}), \forall i \in I, k \in K. \tag{6}$$

$$V_k = \sum_{i \in I} \alpha_{ik} v_i, \forall k \in K \tag{7}$$

$$x_i + w_i \leq W + \Psi(1 - \alpha_{ik}), \forall i \in I, k \in K. \tag{8}$$

$$y_i + l_i \leq L + \Psi(1 - \alpha_{ik}), \forall i \in I, k \in K. \tag{9}$$

$$x_i + w_i \le x_j + \Psi(1 - PL_{ij}), \forall i,j \in I, i \ne j. \tag{10}$$

$$y_i + l_i \le y_j + \Psi(1 - PB_{ij}), \forall i,j \in I, i \ne j. \tag{11}$$

$$PL_{ij} + PL_{ji} + PB_{ij} + PB_{ji} \ge \alpha_{ik} + \alpha_{jk} - 1, \forall i,j \in I, i < j, \forall k \in K. \tag{12}$$

$$\Psi \beta_k \ge \sum_{i \in I} \alpha_{i,k}, \forall k \in K. \tag{13}$$

$$\beta_k \le \sum_{i \in I} \alpha_{ik}, \forall k \in K. \tag{14}$$

$$\beta_{k-1} \ge \beta_k, \forall k \in K \setminus \{1\}. \tag{15}$$

$$\sum_{k \in K} \alpha_{ik} = 1, \forall i \in I. \tag{16}$$

$$x_i, y_i \ge 0, \forall i \in I. \tag{17}$$

$$PL_{ij}, PB_{ij} \in \{0,1\}, \forall i,j \in I. \tag{18}$$

$$\alpha_{ik} \in \{0,1\}, \forall i \in I, k \in K. \tag{19}$$

$$\beta_k \in \{0,1\}, \forall k \in K. \tag{20}$$

The objective function (1) minimizes the makespan, which is the completion time of the last batch. Constraint sets (2) to (4) ensure the calculation of the makespan. Constraint set (5) determines the processing time for batch k. Batch processing time is a function of scanning time, which is proportional to the total volume of parts, and recoating time, which is proportional to the maximum height of the parts in a batch (Kucukkoc, 2019) based on the layer-by-layer printing characteristics of AM. Constraint sets (6) and (7) compute the maximum height and total volume of a given batch k, respectively. Constraint sets (8) to (12) guarantee that a part cannot be placed outside the machine's platform and that two parts cannot overlap if they are assigned to the same batch. Constraint sets (13) to (15) ensure that a part cannot be assigned to an unused batch and that batch k can be processed only if batch $k-1$ is processed first. Constraint set (16) ensures that each part is assigned to only one batch processed by a machine. Constraints (17) to (20) are domain constraints for the variables.

4. Solution methodology

In this section, we first reformulate the MIP model as a set-partitioning master problem (MP). Subsequently, the column generation approach is applied to the linear relaxation of the restricted MP and a novel labelling algorithm is proposed to solve a pricing sub-problem (PSP) and accelerate the column generation process. A two-dimensional rectangular packing algorithm is introduced to verify the packing feasibility of a given batch. Finally, we present a B&P algorithm to obtain an optimal integer solution for the original MIP problem.

4.1. Set partitioning formulation

We utilize the Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960) to reformulate the $C_{\max}$ problem as a set-partitioning MP, as introduced by Rafiee Parsa et al. (2010). In the set-partitioning formulation, $R = (b_1, b_2, \ldots, b_p)$ denotes the set of all feasible columns and each column b_k represents a feasible assignment of parts to a batch. Let the binary variable a_{ik} take a value of one if part i is assigned to column k and a value of zero otherwise. Column b_k is feasible as long as the two-dimensional packing constraints (8) to (12) are satisfied. Let $\alpha_k \in \{0,1\}$ be a binary variable that indicates whether a feasible assignment b_k is selected for batch k. The MP can be formulated as the following integer programming problem:

$$\text{minimize } C_{max} \tag{21}$$

$$\text{s.t.} C_{max} \ge \sum_{k=1}^{P} P_k \alpha_k \tag{22}$$

$$\sum_{k=1}^{p} a_{ik}\alpha_k = 1, \forall i \in I. \tag{23}$$

$$\alpha_k \in \{0,1\}, \forall k \in K. \tag{24}$$

The cost function P_k in constraint (22) is determined by the total volume and maximum height among all parts in batch k (i.e., $P_k = Re \cdot \max_{i\in N}\{h_i a_{ik}\} + Sc \cdot \sum_{i=1}^{n} v_i a_{ik}$, where Re and Sc denote the recoating speed and scanning speed, respectively). Constraint set (23) ensures that each part is assigned to only one batch in the schedule. Constraint set (24) ensures that $\alpha_k, k = 1 \ldots p$ are binary variables.

Practically, the MP cannot be solved directly based on the exponential number of columns (in general, p is prohibitively large). Additionally, most of the associated variables are equal to zero in an optimal solution (Barnhart et al., 1998). Therefore, we rely on solving a restricted MP (RMP) that considers only a subset of the columns of the MP and then add columns on the fly during the column generation process (described later). We consider the following RMP:

$$\text{minimize} \sum_{b\in\tilde{R}} P_a \alpha_b \tag{25}$$

$$\text{s.t.} \sum_{b\in\tilde{R}} a_{ib}\alpha_b = 1, \forall i \in I. \tag{26}$$

$$\alpha_b \geq 0, \forall b \in \tilde{R}. \tag{27}$$

Let $\mu_j \in \mathbb{R}$ be the dual variable associated with constraint set (26) for job $i \in I$. An RMP with small subset $\widetilde{R}$ is relatively easy to solve ($\widetilde{R} \subseteq R$). A column generation method is then invoked to improve the current solution iteratively.

4.2. Column generation

Column generation is used to solve integer programming problems with a large number of decision variables. Starting with some initial columns (a small, but sufficient number of variables to ensure that an original feasible solution exists), we optimize a PSP to determine a new variable instead of explicitly enumerating all possible columns to identify the best variable for the RMP in each iteration. In the case of minimizing the makespan, the most negative reduced cost is identified and a new variable is added to the MP after solving the PSP in each iteration. The column generation procedure terminates when the optimal objective value of the pricing problem is nonnegative, ensuring that no additional columns will improve the result. The optimal solution for the linear relaxation of the original MP is then obtained by solving the current RMP to optimality.

4.2.1. Description of the sub-problem

To guarantee RMP optimality, we must ensure that no variables with a positive reduced cost exist. By solving the relaxed RMP, the dual values $\mu_i, \forall i \in N$ in constraint set (26) are passed to the sub-problem. The variable with the most negative reduced cost is selected by solving the following optimization problem, which is the PSP.

$$\text{minimize } Re \cdot H + Sc \cdot \sum_{i\in I} v_i a_i - \sum_{i\in I} \mu_i a_i \tag{28}$$

$$\text{s.t.} H \geq h_i a_i, \forall i \in I. \tag{29}$$

$$x_i + w_i \leq W + \Psi(1 - a_i), \forall i \in I. \tag{30}$$

$$y_i + l_i \leq L + \Psi(1 - a_i), \forall i \in I. \tag{31}$$

$$x_i + w_i \leq x_j + \Psi(1 - PL_{ij}), \forall i,j \in I, i \neq j. \tag{32}$$

$$y_i + l_i \leq y_j + \Psi(1 - PB_{ij}), \forall i,j \in I, i \neq j. \tag{33}$$

$$PL_{ij} + PL_{ji} + PB_{ij} + PB_{ji} \geq a_i + a_j - 1, \forall i,j \in I, i < j. \tag{34}$$

$$PL_{ij}, PB_{ij} \in \{0,1\}, \forall i,j \in I. \tag{35}$$

$$a_i \in \{0,1\}, \forall i \in I. \tag{36}$$

$$x_i, y_i \geq 0, \forall i \in I. \tag{37}$$

Here, a_i is a binary variable indicating whether part i resides in the new column generated by this PSP. Constraint sets (30) to (34) guarantee that a part cannot be placed outside the machine's platform and that two parts cannot overlap if they are assigned to the same batch. After solving the PSP, a new batch $b \in R \setminus \widetilde{R}$ with the most negative reduced cost is added to $\widetilde{R}$. Otherwise, the column generation procedure terminates and the relaxed optimal solution of the MP is obtained. As long as the objective value of the PSP is negative, iteration continues by repeating the following two steps.

- Solve the relaxed RMP and use the optimal dual values of the RMP to update the cost coefficients of the sub-problem.
- Solve the PSP and add the variable with the most negative reduced cost to the RMP.

4.2.2. *Labelling algorithm*

To solve the PSP, we must transform the model. Because complicated packing constraints are included in the PSP, checking the packing feasibility of a priced-out column is strongly NP-hard. A novel labelling algorithm is proposed in this section to accelerate the column generation process. The goal of this algorithm is twofold. First, to find the column with the minimum reduced cost to be added to the RMP. Second, to identify a pool of columns that satisfy a negative reduced-cost condition so that each column within the pool improves the result.

We will demonstrate that the PSP mentioned above corresponds to a shortest-path problem with resource constraints (Irnich and Desaulniers, 2006). A dynamic programming approach can be used to solve this problem by systematically constructing new paths. A bottom-up approach is adopted by examining the trivial path first and then extending existing paths sequentially in all feasible directions. The labelling algorithm outperforms path enumeration based on a well-designed dominance rule (described later) for discarding non-useful paths. Therefore, only a set of Pareto-optimal paths is maintained during the process. If we set $\beta_i = Sc \cdot v_i - \mu_i, p_i = Re \cdot h_i$ for all $i \in N$, then we can transform the PSP as follows:

$$\text{minimize } \max_{i \in I}(p_i a_i) + \sum_{i \in N} \beta_i a_i \tag{38}$$

$$\text{s.t.} H \geq h_i a_i, \forall i \in I. \tag{39}$$

$$x_i + w_i \leq W + \Psi(1 - a_i), \forall i \in I. \tag{40}$$

$$y_i + l_i \leq L + \Psi(1 - a_i), \forall i \in I. \tag{41}$$

$$x_i + w_i \leq x_j + \Psi\left(1 - PL_{ij}\right), \forall i,j \in I, i \neq j. \tag{42}$$

$$y_i + l_i \leq y_j + \Psi\left(1 - PB_{ij}\right), \forall i,j \in I, i \neq j. \tag{43}$$

$$PL_{ij} + PL_{ji} + PB_{ij} + PB_{ji} \geq a_i + a_j - 1, \forall i,j \in I, i < j. \tag{44}$$

$$a_i \in \{0,1\}, \forall i \in I. \tag{45}$$

$$x_i, y_i \geq 0, \forall i \in I. \tag{46}$$

$$PL_{ij}, PB_{ij} \in \{0,1\}, \forall i,j \in I. \tag{47}$$

Following this transformation, the printing time calculation formula is split into two parts. The first part is determined by the maximum height and the second part can be grouped with the dual term. This transformation makes it possible to use the labelling algorithm presented in Algorithm 1. Let V be a set of vertices corresponding to parts $i = 1, \ldots, n$, $n = |N|$ that are topologically sorted in a non-increasing order of their heights. Vertex i is linked to vertex j by edge (i,j) if and only if $i < j$ and E denotes the set of all edges. The reduced cost and sequence of visited vertices are denoted as C^i and L^i, respectively, for each

$i \in V$. The algorithm runs for n iterations after initialization. At iteration i, node i is selected, all paths from this node are extended to j if $j > i$ and $L^j \leftarrow L^i \cup i$ satisfies the two-dimensional packing constraints mentioned above, and label $X^j_{l'}$ is assigned to a new path l' with updated resources. A pool of paths is returned at the end of the algorithm and the path with the greatest negative reduced cost is added to the RMP. However, it is difficult to implement a packing subroutine in the column generation procedure. Instead of checking packing feasibility for a given batch every time the path is extended to a new node, we relax the two-dimensional rectangular packing constraints to the one-dimensional knapsack constraints and solve the following relaxed PSP.

$$\text{minimize } \max_{i \in I}(p_i a_i) + \sum_{i \in N} \beta_i a_i \tag{48}$$

$$\text{s.t.} \sum_{i \in I} s_i a_i \le \delta B, \tag{49}$$

$$a_i \in \{0,1\}, \forall i \in N. \tag{50}$$

The bottom area of the machine platform is denoted as $B, B = WL$. At each iteration, all paths from node i are extended to j if $j > i$ and $S^i + s_j \le \delta B, \delta \le 1$, and label $X^j_{l'}$ is assigned to a new path l' with the updated resources. The control parameter δ is used to ensure that a priced-out path that satisfies the two-dimensional packing constraints is more likely to appear in the early stages. We use an open-space-based packing algorithm (see Subsection 4.2.3) to check the packing feasibility of a set of potentially feasible paths with negative reduced costs at the end of the labelling algorithm (see Algorithm 1).

As a result, the efficiency of the labelling algorithm heavily relies on its ability to eliminate non-promising labels. Otherwise, the labelling algorithm will have no advantage compared to the enumeration method. This property gives rise to dominance rules that allow suboptimal paths to be removed safely during the search. Here, we refer to the definitions introduced by Zhang et al. (2022).

Definition 1. Given two items (w_i, l_i) and (w_j, l_j), if $w_i \le w_j$ and $l_i \le l_j$, then we denote $(w_i, l_i) \sqsubseteq (w_j, l_j)$.

Definition 2. Given two sets of items L_i and L_j, we say that L_i is covered by L_j if there is a one-to-one mapping such that for any item $k \in L_i$, there is an item $k' \in L_j$ such that $k \sqsubseteq k'$. Coverage is denoted as $L_i \sqsubseteq L_j$.

Based the definitions above, the dominance rule can be defined as follows.

Theorem 1. *Label X^i_l is dominated by label $X^i_{l'}$ (denoted as $X^i_l \prec X^i_{l'}$) if the following conditions are satisfied*:

(1) $C^i_{l'} \le C^i_l$;
(2) $L^i_{l'} \sqsubseteq L^i_l$.

Although Theorem 1 can be treated as solving a maximum bipartite matching problem, which is solvable in polynomial time (Kleinberg and Tardos, 2006), the dominance rule is predictably weak (Zhang et al., 2022). Therefore, designing a new approximate dominance rule is necessary. We consider the total area (denoted as S^i) as one of the label attributes and the total area of a path is defined by the summation of the bottom area of each part included in L^i (i.e., $S^i = \sum_{j \in L^i} s_j, s_j = w_j l_j$). Theorem 2 defines an approximate rule.

Theorem 2. *Label X^i_l is approximately dominated by label $X^i_{l'}$ (denoted as $X^i_l \ll X^i_{l'}$) if the following conditions are satisfied*:

(1) $C^i_{l'} \le C^i_l$;
(2) $\gamma S^i_{l'} \le S^i_l$.

Based on Theorem 2, the new dominance rule embedded in the algorithm can be summarized as follows. A label $X^i_{l'}$ approximately dominates X^i_l if and only if $C^i_{l'} \le C^i_l$ and $\gamma S^i_{l'} \le S^i_l, \gamma \ge 1$. A control parameter γ is introduced to manipulate the tightness of the dominance rule because Theorem 2 does not guarantee perfect dominance. We start with $\gamma = 1$ so that more labels can dominate in the early stages. Every time a new label l' is created at node i, we compare the labelled path to all paths ending in i according to the approximate dominance rule. If a path is dominated, it is discarded to accelerate the labelling algorithm. When there are no improved columns with the current γ, this parameter is tuned to weaken the dominance condition so that more labels can be released during the search process.

Algorithm 1. Labelling algorithm

```
Algorithm 1. Labelling algorithm
Input: A directed graph $G = (V, E)$
Output: A set of feasible paths with negative reduced cost
1:  for $i \in N$ do
2:      Initialize $C^i = p_i + \beta_i, S^i = s_i, L[l] = [i]$
3:      Path $l$ ← New label $X_l^i$
4:      Add path $l$ to the column pool ($Col_{pool}$)
5:  end
6:  for $i \in N$ do
7:      Select all paths ended in the node $i$
8:      for $j \in i.adj$ do                          ◆ Adjacent node of $i$
9:          if $S^i + s_j \le B$ then
10:             $L[l'] = L[i] \cup j, C^j = C^i + \beta_j, S^j = S^i + s_j$
12:             Path $l'$ ← New label $X_{l'}^j$
12:             if exists label $X_l^j, X_l^j \ll X_{l'}^j$ then
13:                 Discard path $l'$
14:             else
15:                 Add path $l'$ to the $Col_{pool}$
16:             end
17:         end
18:     end
19: end
20: for $path \in Col_{pool}$ do
21:     $Batch$ ← parts included in $path$
22:     if $Packing(Batch) == False$ then             ◆ Check packing feasibility of the batch
23:         Discard $path$ from $Col_{pool}$
24:     end
25: end
```

4.2.3. Open-space-based heuristic packing algorithm

In this section, we verify the packing feasibility of a given batch by executing a two-dimensional rectangular packing algorithm. Wei et al. (2018) proposed an open-space-based heuristic packing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints and we adopted this approach in our research (see Algorithm 2). The packing algorithm takes a sequence of ordered parts R as input and attempts to pack all parts into the machine platform. The maximum open space (MOS) M is defined by the following four attributes.

- $M.x, M.y$: coordinates of the bottom-left corner of M
- $M.w, M.l$: width and length of M

The initial machine platform is represented by a MOS located at point $(0, 0)$ with dimensions of $W \times L$. As long as the algorithm continues, the open space with a corner farthest toward the bottom-left corner is selected as the best-fitted area (based on a scoring rule) and placed at the bottom-left corner of a MOS. The open-space list is then updated. If a feasible packing solution is found, a constant value of *True* is returned. Otherwise, a constant value of *False* is returned.

Algorithm 2. Packing algorithm

```
Algorithm 2. Packing algorithm
Input: Parts list $R$
Output: Packing feasibility, placement coordinates $(x_i, y_i)$ of part $i \in R$
 1: Initialize open space $M.x = 0, M.y = 0, M.w = W, M.l = L$
 2: $Spacelist = \{M\}$, place the first part $r$ at $M$
 3: Update the open space list $SpaceList$ and remove $r$ from $R$
 4: while $R \neq \emptyset$ & $SpaceList \neq \emptyset$ do
 5:     $M_{BL}$ = space with the most bottom-left corner
 6:     Remove $M_{BL}$ from $SpaceList$
 7:     Find $r$ that is best-fit into $M$
 8:     if such $r$ is found then
 9:         Place $r$ at $M$
10:         Update the open space list $SpaceList$ and remove $r$ from $R$
11:     end
12: end
```

4.3. Branch and price

Solving the linear programming (LP) relaxation of the MP using the aforementioned column generation procedure may not guarantee an integer optimal solution and simply optimizing a MIP model of the MP over existing columns will not ensure an optimal solution to the original problem. Therefore, it is necessary to branch the non-integer variables and generate additional columns along the search tree. In this section, we introduce a B&P algorithm to achieve optimality.

4.3.1. Description of the algorithm

B&P has been used to solve a wide variety of combinatorial optimization problems, including vehicle routing problems (Taş et al., 2014; Reihaneh and Ghoniem, 2019), machine scheduling problems (Rafiee Parsa et al., 2010; Muter, 2020), and facility location problems (Klose and Görtz, 2007; Ni et al., 2021). The B&P method features column generation embedded within a branch-and-bound framework.

In a B&P tree, the root node is a linear relaxation of the original MP. After deploying the column generation method for the root node, we set the initial lower bound as the optimal objective value for the LP relaxation of the root node model. Optimizing the MIP model of the MP over existing columns (set-covering problem) determines the initial upper bound. During the B&P process, we maintain a global upper bound to represent the best feasible solution obtained thus far, which is also known as the incumbent. In each B&P iteration, the RMP of node i inherits all the columns from its parent node, except those that must be deleted according to the branching rule (described later). The lower bound of node i is obtained by solving the RMP using the column generation approach. Instead of exploring the entire B&P tree, pruning strategies can be used to discard valueless branches.

Specifically, three cases lead to a node i being pruned.

- After branching, a restriction is added to the current node, which makes LP relaxation infeasible.
- The lower bound of node i is found, but its value is greater than the global upper bound (current best feasible solution). There is no need to explore this subtree because the optimal objective value of this node represents the best lower bound its offspring can possibly obtain, so the nodes in this subtree can never generate better integer solutions.
- An optimal integer solution LB_i is found and is smaller than the current incumbent. In this case, node i becomes the best integer node. We update the global upper bound as $UB \leftarrow LB_i$.

Starting from the root node, we use different rules to select nodes from the queue and continue exploring as long as the queue is not empty. For example, the depth-first-search rule (DFS), breadth-first-search rule (BFS), and best-bound-first-search rule (BBFS) can be considered. The DFS and BFS rules are common strategies for searching a tree data structure and the BBFS rule can be obtained by using a priority queue that sorts nodes according to their lower bound. For example, suppose that we have solved the linear relaxation of a node using the column generation method. If an integer solution is obtained, then the node is pruned to determine whether the solution is superior or inferior to the incumbent. The only difference is that if the newly generated feasible solution is superior to the incumbent, then we set this node as the new incumbent and update the upper bound UB to the new objective value of the MIP model. If the solution is fractional, which is often the case, we must branch the node and create two child nodes following the branching rule. This branching strategy is discussed in the following section.

4.3.2. Branching strategy

As mentioned previously, simply solving the linear relaxation of a given B&P node using the column generation technique does not guarantee an integer solution. When a fractional solution appears, it is necessary to branch the node and generate subtrees to ensure an optimal integer solution by repeatedly performing column generation on the child nodes. The keys to devising a branching rule are twofold: the current fractional solution must be excluded from the subtree and tractability must be guaranteed in the sub-problem (Vance et al., 1994).

Ryan and Foster (1981) proposed a well-known branching strategy based on the following proposition. Rafiee Parsa et al. (2010) followed the same strategy in their work and we adopted the same approach in our B&P algorithm.

Proposition 1. *If A is a zero-one matrix and a basic solution to $AX = 1$ is fractional, meaning at least one of the components of X is fractional, then there are two rows –l and m– in the matrix of coefficients of the MP such that* $0 < \sum_{k:a_{lk}=1,a_{mk}=1} \alpha_k < 1$.

Consequently, the pair l, m yields the following pair of branching constraints

$$\sum_{k:a_{lk}=1,a_{mk}=1} \alpha_k = 1 \text{ and } \sum_{k:a_{lk}=1,a_{mk}=1} \alpha_k = 0$$

The interpretation of this rule is natural, as we can see that rows l and m must be covered by the same column on the left branch and by different columns on the right branch. Each row corresponds to one part. Therefore, we call this branching strategy "branching in part pairs." The B&P algorithm with this branching strategy (BP_{bpp}) must terminate after a finite number of branches because there is only a finite number of pairs of rows.

For comparison, we consider another branching rule imposed on the B&P algorithm, namely BP_{bsv}, which is conducted on a single variable in the set partitioning formulation. Specifically, suppose we have solved the relaxation model of a given node using a column generation procedure. If the node cannot be pruned, then we can select a variable in the non-integer solution such that $\alpha_k = \arg\min_{k\in R}\{\alpha_k - 0.5\}$, implying that the pattern variable with the fractional part closest to 0.5, say α_k, is selected as the branching variable. Then, we generate two new MIPs, where $\alpha_k \leq 0$ is imposed on the left branch and $\alpha_k \geq 1.0$ is imposed on the right branch. Branching imposes new constraints on the sub-problems. However, the newly generated columns may violate the branching constraints. As suggested by Degraeve and Schrage (1999), given a forbidden batch $b \subseteq N$, the anticycling constraint is

$$\sum_{i\in S}\alpha_i - \sum_{i\in N\setminus b} \alpha_i \leq |b| - 1$$

The same inequality was also used by van Zon et al. (2021) to solve the cost allocation problem in a joint network vehicle routing game. The anticycling constraint ensures that no column excluded by the branching rule will be generated again by the PSP.

After branching, additional constraints are imposed on each MIP. Therefore, it is necessary to use the column generation technique to generate valuable columns or prove LP optimality for each sub-problem on the way to an optimal solution. The imposed constraints are encoded in the labelling algorithm to prevent the generation of columns that have been invalidated with respect to the branching rule. The idea behind this feature is that we select a suitable branching strategy to restrict sub-problems. Therefore, the solution of a parent node can be considered as the better solution between two more constrained sub-problems.

5. Computational experiments

We conducted extensive experiments to evaluate the performance of our novel B&P algorithm systematically. Our experiments were conducted on a PC with an Intel Core i5-7500 3.40 GHz CPU with 8 GB RAM. We first introduce the numerical settings of our study based on a parts dataset from a 3D printing factory in Shanghai and then present the computational results of our experiments. We also shed light on different branching strategies and node-selection rules, which are important components of the B&P algorithm.

5.1. Numerical settings

The parts dataset was provided by our 3D printing industry partner. For each part, information on several attributes was collected, including the width, length, height, and volume. Table 1 presents several statistics for the attributes of the parts. Each instance that we solved was characterized by a number of parts n randomly selected from the same dataset. Our numerical experiments are based on the same parameter settings and Table 2 presents the width and length of the machine platform, as well as the scanning speed and recoating speed of the machine. The classes we considered differed in terms of part number $n \in \{8, 10, 12, 20, 40, 60\}$ and ten instances for each class were sampled from the dataset.

Table 1
Statistics on the part attributes.

Statistic / Attribute	Mean	Minimum	Median	Maximum
Width (cm)	26.6	2.8	24.5	58.1
Length (cm)	23.8	1.8	20.0	58.1
Height (cm)	19.4	1.0	16.4	59.0
Volume (cm^3)	3700.0	10.0	1867.5	47032.4

5.2. Computational results

The performance of the B&P algorithm was compared to that of the MIP model introduced in Section 3. The initial MIP model was solved using a popular commercial solver (Gurobi 9.0.0). The solver terminated if one of the following conditions was satisfied: 1) the optimal solution was found by Gurobi of 2) the computing time reached a limit of 3600 s. For each instance, the approximate gap was calculated as $gap_{relative} = 100*(C_{bp} - C_{gurobi})/C_{gurobi}$, where C_{gurobi} denotes the best feasible solution found by Gurobi and C_{bp} denotes the best feasible solution found by the B&P algorithm. The notations used in the tables are summarized below.

- $Av.gap_{relative}$: Average approximate gap calculated for instances where the B&P solution is compared to the best solution found by Gurobi in 3600 s.
- $Max.gap_{relative}$: Maximum approximate gap calculated for instances where the B&P solution is compared to the best solution found by Gurobi in 3600 s.
- CPU Time/s: Average computation time in seconds calculated as the average running time of ten numerical instances.
- N_{opt}^{gurobi}: Number of instances for which an optimal solution was found by Gurobi in 3600 s.

Table 3 compares the performance measures of B&P algorithms with two different branching strategies: branching on part pairs (BP_{bpp}) and branching on single variables (BP_{bsv}). The Gurobi counterpart is used as a benchmark. According to Table 3, both algorithms can generate similar results or the same results as Gurobi for all instances when the number of parts $n \in \{10, 20, 40\}$. However, although BP_{bpp} performs slightly worse than BP_{bsv} in terms of solution quality for instances where Gurobi cannot solve to optimality within a time limit of 3600 s, the computing time of BP_{bpp} is significantly shorter than that of BP_{bsv}, especially as the number of parts increases. This is a result of the extreme asymmetry incurred by the latter branching strategy. Branching on a single variable may require significant effort to ensure the optimality of the algorithm based on the presence of numerous feasible batch variables. Therefore, we selected the branching on part pairs strategy for the following numerical experiments.

We were also curious about the impact of different node selection rules. Therefore, we considered the three common rules (DFS, BFS, BBFS) described in Section 4.3.1 in combination with our B&P algorithm. Tables 4 and 5 list the performance measures of the B&P algorithm with these three search rules and those of the Gurobi optimizer. We consider small instances in which optimal solutions can be verified by Gurobi within a reasonable time and large instances in which optimal solutions cannot be verified within a time limit of 3600 s. For small instances (see Table 4), our B&P algorithm can generate the same results for almost all instances in a short time, regardless of the node selection rules. For the medium and large instances (see Table 5), our B&P algorithm can generate similar results to its Gurobi counterpart in a practical amount of computation time ($Av.gap_{relative} < 0.6\%$) and even outperforms Gurobi in terms of solution quality and running time for many large instances ($Av.gap_{relative} < -0.6\%$). It should be noted that BP^{BBFS} uses only two-thirds of the running time on average compared to the B&P algorithms with the other two search rules when the number of parts is $n = 60$. This may be because the seemingly myopic rule that always selects the node with the minimum lower bound makes it easier to find better feasible solutions, which reduces the burden of exploring and pruning.

In summary, for small instances in which Gurobi solves optimality within the time limit of 3600 s, our B&P algorithm can generate the same or similar results for almost all instances. For medium and large instances, where the optimal solution cannot be guaranteed by Gurobi within a time limit of 3600 s, our B&P algorithm can generate a solution similar to the best feasible solution found by Gurobi in a reasonable time, even beating Gurobi in terms of solution quality and running time on many large instances. We also examined two important components of the B&P algorithm. First, the branching strategy plays

Table 2
Parameter values of the machine.

Parameter	Signification	Value
W	Width of machine platform (cm)	60
L	Length of machine platform (cm)	60
Sc	Scanning speed (h/cm^3)	0.003
Re	Recoating speed (h/cm)	0.07

Table 3
Comparisons of B&P algorithms with different branching rules.

n	BP_{bpp}			BP_{bsv}			Gurobi	
	$Av.gap_{relative}$(%)	$Max.gap_{relative}$ (%)	CPU Time/s	$Av.gap_{relative}$ (%)	$Max.gap_{relative}$ (%)	CPU Time/s	N_{opt}^{gurobi}	CPU Time/s
10	0.0	0.0	0.2	0.0	0.0	0.8	10	5.2
20	0.5	1.5	1.6	0.3	0.8	31.6	0	3600.0
40	0.6	1.3	55.3	0.1	0.8	795.3	0	3600.0

Table 4
Small-instance quality comparisons of B&P algorithms.

n	BP^{DFS}		BP^{BFS}		BP^{BBFS}		Gurobi	
	$Av.gap_{relative}$(%)	CPU Time/s	$Av.gap_{relative}$(%)	CPU Time/s	$Av.gap_{relative}$(%)	CPU Time/s	N_{opt}^{gurobi}	CPU Time/s
8	0.00	0.11	0.00	0.11	0.00	0.11	10	0.42
10	0.00	0.17	0.00	0.18	0.00	0.17	10	5.19
12	0.29	0.33	0.29	0.34	0.29	0.30	10	166.52

Table 5
Medium-to-large-instance quality comparisons of B&P algorithms.

n	BP^{DFS}		BP^{BFS}		BP^{BBFS}		Gurobi	
	$Av.gap_{relative}$(%)	CPU Time/s	$Av.gap_{relative}$(%)	CPU Time/s	$Av.gap_{relative}$(%)	CPU Time/s	N_{opt}^{gurobi}	CPU Time/s
20	0.51	1.61	0.50	1.36	0.51	1.38	0	3600
40	0.58	55.27	0.52	67.32	0.57	58.14	0	3600
60	−0.64	284.44	−0.63	290.58	−0.65	195.09	0	3600

a significant role in the performance of the proposed B&P algorithm. For small and medium instances, the node selection rule has little impact on our B&P algorithm. However, the average running time of the B&P algorithm with the BBFS rule was much shorter for large instances. Overall, the B&P algorithm provides satisfactory performance in terms of average running time and solution quality.

6. Conclusions

Motivated by our AM industry partner, we considered the problem of scheduling parts with multiple attributes on a single BPM to minimize the makespan in a make-to-order environment. We developed a B&P algorithm for this practical variant of the BPM scheduling problem with two-dimensional packing. Our B&P algorithm embeds a column generation method into a branch-and-bound framework and utilizes a novel labeling algorithm to accelerate the column generation process. In a B&P tree, each node is the linear relaxation of a set-partitioning MP and column generation is used to select columns to add to the MP or prove the optimality of the node.

Based on real data from a 3D printing factory, numerical experiments indicated that our B&P algorithm yields satisfactory results compared to the state-of-the-art Gurobi optimizer. Specifically, the proposed B&P algorithm can generate similar or identical results compared to Gurobi in a reasonable amount of computation time for most real-world instances. For some large instances, our novel technique eve produces better solutions in terms of solution quality and running time compared to Gurobi within a time limit of 3600 s.

We propose two directions to extend this research. First, our study focused on scheduling parts on a single AM machine and printing time estimation. More practical considerations, including parallel AM machine scheduling problems, could also be considered. Another potential direction is to consider scheduling parts with release times in a dynamic environment.

Declaration of competing interest

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

Acknowledgement

This research was supported by the National Nature Science Foundation of China (NSFC) with grant Nos: 72091215/72091210, 71921001 and 72022018, and Youth Innovation Promotion Association (Grant No. 2021454).

References

Azizoglu, M., & Webster, S. (2001). Scheduling a batch processing machine with incompatible job families. *J. Art. Comput. Ind. Eng., 39*(3), 325–335.
Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., & Vance, P. H. (1998). Branch-and-Price: column generation for solving huge integer programs. *Oper. Res., 46*(3), 316–329.
Beldar, P., & Costa, A. (2018). Single machine batch processing problem with release dates to minimize total completion time. *J. Article. Int. J. Ind. Eng. Comput. , 9*, 331–348.
Che, Y., Hu, K., Zhang, Z., & Lim, A. (2021). Machine scheduling with orientation selection and two-dimensional packing for additive manufacturing. *J. Article. Comput. Oper. Res., 130*, Article 105245.
Chen, Z., & Powell, W. B. (1999). Solving parallel machine scheduling problems by column generation. *J. Article. INFORMS J. Comput., 11*(1), 78–94.
Damodaran, P., Manjeshwar, P. K., & Srihari, K. (2006). Minimizing makespan on a batch-processing machine with non-identical job sizes using genetic algorithms. *Int. J. Prod. Econ., 103*, 882–891.
Dantzig, G. B., & Wolfe, P. (1960). Decomposition principle for linear programs. Journal article. *Oper. Res., 8*(1), 101–111.
Degraeve, Z., & Peeters, M. (2003). Optimal integer solutions to industrial cutting-stock problems: Part 2, benchmark results. *J. Article. INFORMS J. Comput., 15*(1), 58–81.
Degraeve, Z., & Schrage, L. (1999). Optimal integer solutions to industrial cutting stock problems. *Inf. J. Comput., 11*(4), 406–419.
Dobson, G., & Nambimadom, R. S. (2001). The batch loading and scheduling problem. *J. Article. Oper. Res., 49*(1), 52–65.
Dong, L., Shi, D., & Zhang, F. (2020). *3D Printing and Product Assortment Strategy*. Available at: SSRN 2847731.
Faiz, T. I., Vogiatzis, C., & Noor-E-Alam, M. (2019). A column generation algorithm for vehicle scheduling and routing problems. *Comput. Ind. Eng., 130*, 222–236. Elsevier.
Fanjul-Peyro, L., Ruiz, R., & Perea, F. (2019). Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. *J. Article. Comput. Oper. Res., 101*, 173–182.
Fowler, J. W., & Lars Mönch, L. (2022). A survey of scheduling with parallel batch (P-batch) processing. *J. Article. Eur. J. Oper. Res., 298*, 1–24.
Ghodratnama, A., Rabbani, M., Tavakkoli-Moghaddam, R., & Baboli, A. (2010). Solving a single-machine scheduling problem with maintenance, job deterioration and learning effect by simulated annealing. *J. Manuf. Syst., 29*(1), 1–9. Elsevier.
Irnich, S., & Desaulniers, G. (2006). In *Shortest Path Problems with Resource Constraints. Book Section* (pp. 33–65).
Jin, F., Song, S., & Wu, C. (2009). A simulated annealing algorithm for single machine scheduling problems with family setups. *Comput. Oper. Res., 36*(7). Elsevier: 2133–8.
Kashan, A. H., Karimi, B., & Jolai, F. (2006). Effective hybrid genetic algorithm for minimizing makespan on a single-batch-processing machine with non-identical job sizes. *J Article. Int. J. Prod. Res., 44*(12), 2337–2360.
Kleinberg, J., & Tardos, E. (2006). *Algorithm Design*. Pearson Education India.
Klose, A., & Görtz, S. (2007). A branch-and-price algorithm for the capacitated facility location problem. *Eur. J. Oper. Res., 179*(3), 1109–1125.
Kucukkoc, I. (2019). MILP models to minimise makespan in additive manufacturing machine scheduling problems. *Comput. Oper. Res., 105*, 58–67.
Larrañeta, E., Dominguez-Robles, J., & Lamprou, D. A. (2020). Additive manufacturing can assist in the fight against covid-19 and other pandemics and impact on the global supply chain. *3D Print. Addit. Manuf., 7*(3), 100–103.
Li, S., Li, G., Wang, X., & Liu, Q. (2005). Minimizing makespan on a single batching machine with release times and non-identical job sizes. *Journal Article. Operations Research Letters, 33*(2), 157–164.
Liu, L. L. (2013). Online scheduling with incompatible job families on batch processing machine (S). In *Applied Mechanics and Materials* (vol. 345, pp. 408–411). Trans Tech Publ.
Meng, Y., & Tang, L. (2010). A tabu search heuristic to solve the scheduling problem for a batch-processing machine with non-identical job sizes. *2010 Int. Conf. Log. Syst. Int. Manag. (Iclsim), 3*, 1703–1707.
Muter, İ. (2020). Exact algorithms to minimize makespan on single and parallel batch processing machines. *J. Article. Eur. J. Oper. Res., 285*(2), 470–483.
Ni, W., Shu, J., Song, M., Xu, D., & Zhang, K. (2021). A branch-and-price algorithm for facility location with general facility cost functions. *Inf. J. Comput., 33*(1), 86–104.
Ozturk, O., Espinouse, M.-L., Mascolo, M. D., & Gouin, A. (2012). Makespan minimisation on parallel batch processing machines with non-identical job sizes and release dates. Journal article. *Int. J. Prod. Res., 50*(20), 6022–6035.
Rafiee Parsa, N., Karimi, B., & Husseinzadeh Kashan, A. (2010). A branch and price algorithm to minimize makespan on a single batch processing machine with non-identical job sizes. *J. Article. Comput. Oper. Res., 37*(10), 1720–1730.
Rafiee Parsa, N., Karimi, B., & Moattar Husseini, S. M. (2017). Exact and heuristic algorithms for the just-in-time scheduling problem in a batch processing system. *J. Article. Comput. Oper. Res., 80*, 173–183.
Rafiee Parsa, N., Keshavarz, T., Karimi, B., & Moattar Husseini, S. M. (2021). A hybrid neural network approach to minimize total completion time on a single batch processing machine. *J. Article. Int. Trans. Oper. Res., 28*(5), 2867–2899.
Reihaneh, M., & Ghoniem, A. (2019). A branch-and-price algorithm for a vehicle routing with demand allocation problem. *Eur. J. Oper. Res., 272*(2), 523–538.
Ryan, D. M., & Foster, B. A. (1981). An integer programming approach to scheduling. *Comput. schedul. publ. transport. urban passenger veh. crew sch.*, 269–280.
Song, J.-S., & Zhang, Y. (2020). Stock or print? Impact of 3-d printing on spare parts logistics. *Manag. Sci., 66*(9), 3860–3878.
Tanaka, S., Fujikuma, S., & Araki, M. (2009). An exact algorithm for single-machine scheduling without machine idle time. *Journal Article. J. Schedul., 12*(6), 575–593.
Taş, D., Gendreau, M., Dellaert, N., van Woensel, T., & de Kok, A. G. (2014). Vehicle routing with soft time windows and stochastic travel times: a column generation and branch-and-price solution approach. *Eur. J. Oper. Res., 236*(3), 789–799.
Trindade, R. S., de Araújo, O. C. B., & Fampa, M. (2021). arc-flow approach for single batch-processing machine scheduling. *J. Article. Comput. Oper. Res., 134*, Article 105394.
Uzsoy, R. (1994). Scheduling a single batch processing machine with non-identical job sizes. *J Article. Int. J. Prod. Res., 32*(7), 1615–1635.
Uzsoy, R. (1995). Scheduling batch processing machines with incompatible job families. *J Article. Int. J. Prod. Res., 33*(10), 2685–2708.
van Zon, M., Spliet, R., & van den Heuvel, W. (2021). The joint network vehicle routing game. *Transport. Sci., 55*(1), 179–195. INFORMS.
Vance, P. H., Barnhart, C., Johnson, E. L., & Nemhauser, G. L. (1994). Solving binary cutting stock problems by column generation and branch-and-bound. *Journal Article. Comput. Optimiz. Appl., 3*(2), 111–130.
Wang, C.-S., & Uzsoy, R. (2002). A genetic algorithm to minimize maximum lateness on a batch processing machine. *J. Article. Comput. Oper. Res., 29*(12), 1621–1640.
Wei, L., Zhang, Z., Zhang, D., & Leung, S. C. H. (2018). A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints. *J. Article. Eur. J. Oper. Res.*
Xu, R., Chen, H., & Li, X. (2012). Makespan minimization on single batch-processing machine via ant colony optimization. *J. Article. Comput. Oper. Res., 39*(3), 582–593.
Zarook, Y., Rezaeian, J., Tavakkoli-Moghaddam, R., Mahdavi, I., & Javadian, N. (2015). Minimization of makespan for the single batch-processing machine scheduling problem with considering aging effect and multi-maintenance activities. *Journal Article. The Int. J. Adv. Manuf. Technol., 76*(9), 1879–1892.

Zhang, J., Yao, X., & Li, Y. (2020). Improved evolutionary algorithm for parallel batch processing machine scheduling in additive manufacturing. *J Article. Int. J. Prod. Res., 58*(8), 2263–2282.

Zhang, X., Chen, L., M, & Langevin, A. (2022). Learning-based branch-and-price algorithms for the vehicle routing problem with time windows and two-dimensional loading constraints. *J. Article. INFORMS J. Comput.*

Zheng, S., Xie, N., & Wu, Q. (2021). Single batch machine scheduling with dual setup times for autoclave molding manufacturing. *J. Article. Comput. Oper. Res., 133*, Article 105381.