Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/caor

Machine scheduling with orientation selection and two-dimensional packing for additive manufacturing

Yuxin Che [a,1], Kanxin Hu [a,*,1], Zhenzhen Zhang [b], Andrew Lim [a]

[a] Department of Industrial Systems Engineering and Management, National University of Singapore, Singapore 117576, Singapore
[b] School of Economics and Management, Tongji University, Shanghai 200092, China

A R T I C L E I N F O

Article history:
Received 12 May 2020
Revised 4 February 2021
Accepted 4 February 2021
Available online 11 February 2021

Keywords:
scheduling
Additive manufacturing
Unrelated parallel batch machines
Part orientation selection
Two-dimensional bin packing
Heuristics

A B S T R A C T

In recent years, additive manufacturing (AM) gains increasing attention in manufacturing industries due to the growing demands, and the corresponding machine scheduling problems also attract many researchers. In this paper, we study a new unrelated parallel batch processing machine scheduling problem arose in AM, which requires to simultaneously assign parts to batches, determine the orientation of parts, pack the parts to the two-dimensional surface, and allocate the batches to machines. It is the first work to consider the orientation selection for parts in machine scheduling with the objective to minimize the makespan. To solve this problem, we first present a mixed integer linear programming model. Then, a simulated annealing algorithm with designed packing strategies based on the skyline representation of packing pattern is developed. Moreover, data structure Trie is introduced to accelerate the whole procedure and four post-optimization methods are designed to further refine solutions. Finally, a comprehensive computational study is conducted. The efficiency of our heuristic algorithms is verified and the best packing strategy for this problem is identified. The advantage of considering multiple orientations of parts in machine scheduling is demonstrated by comparisons with scenarios of fixed orientation.

1. Introduction

This paper is motivated by production planning in the booming domain of manufacturing industrial—additive manufacturing (AM), also known as 3D printing. AM is the method to build parts from 3D digital model by forming materials layer by layer with effective technologies such as selective laser melting, laser engineered shaping, and electron beam melting (Guo and Leu, 2013). With these technologies, AM provides quite a lot of advantages such as design flexibility, model complexity, manufacturing efficiency, etc. According to Pour et al. (2016), AM is promising to have a significant impact on manufacturing supply chains and systems. Moreover, its total market is predicted to rise nearly to $12 billion in 2025 (Vicari, 2015).

Although AM has received extensive attentions from researchers in recent years, most of studies focus on the AM process itself, such as material renovation and upgrade of machine components (Kucukkoc, 2019). As AM has been widely used and many AM service factories have appeared in the commercial market, the requirements of time and quality also need to be addressed. Hence, the efficiency of the machine scheduling part in the AM production becomes increasingly important, which is the motivation of this study.

We focus on the selective laser melting (SLM) of AM process. In SLM, the designated parts are formed by material powder, which is melted by a laser beam, layer by layer, as shown in Fig. 1. First, one recoater blade spreads a thin layer of powder over machine's building platform. Second, a moving laser scans the specified area to melt powder, and then one layer of the part is formed. Third, the building platform moves down and steps 1 and 2 are executed again. By repeating these three steps, the parts can be fully formed eventually.

The detailed AM scheduling problem is shown in Fig. 2, which includes two stages: parts assignment and batches allocation. The parts are first grouped into several batches, while the second stage is to allocate the batches to different machines with respect to the machine capacity. Note that, the orientations of parts are determined explicitly in the first stage. For each part, it has several orientation candidates corresponding to different build directions in SLM machine. Fig. 3 shows one example of orientation candidates. Different orientations lead to different building heights, support structures and production areas, which finally affect the building cost and time. Moreover, several heterogeneous AM machines are available and each can produce a batch of parts simultaneously. More specifically, every machine has its own specifications, such

* Corresponding author.
E-mail address: kxhu125@gmail.com (K. Hu).
[1] Yuxin Che and Kanxin Hu contribute equally to this paper.

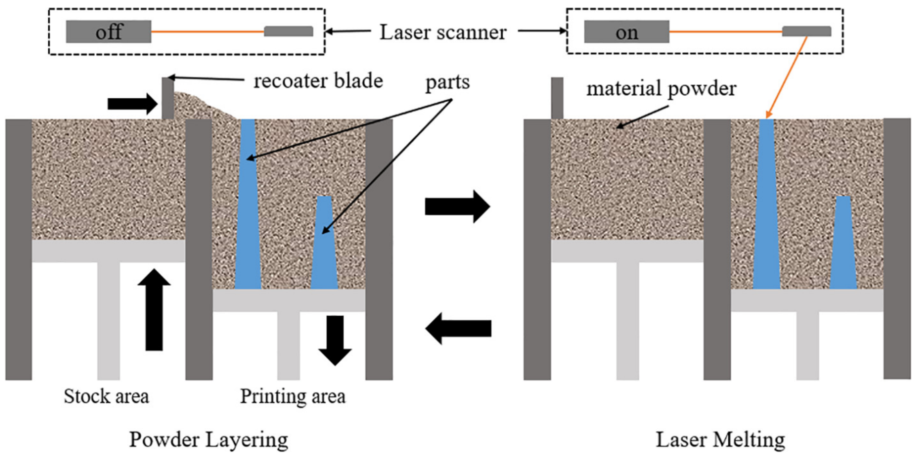

Fig. 1. The process of SLM. Powder Layering Process: the recoater blade moves material powder from stock area to printing area and spreads a thin layer of powder over printing area. Laser Melting Process: laser scanner is turned on and scans the specified area to melt powder to form a structural layer of the part.

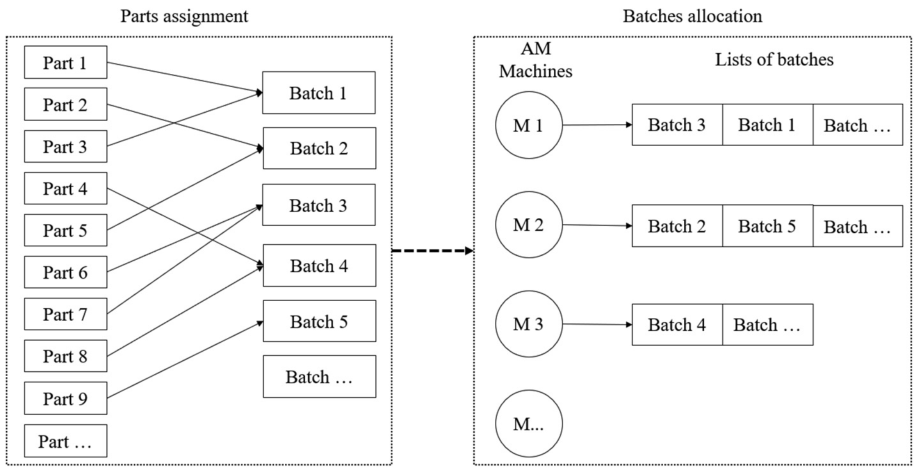

Fig. 2. AM scheduling process. First, parts are assigned to different batches. Second, batches are allocated to different additive machines.

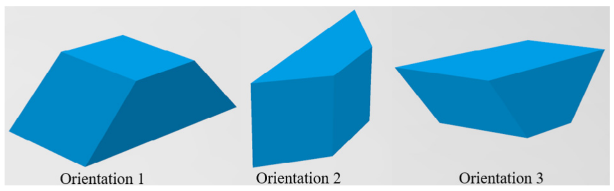

Fig. 3. An example of orientation candidates for one part.

as setup time, production speed, and machine capacity. Before starting the production process, a series of time-consuming operations need be performed to set up the machines, such as model data uploading, parameters setting and material filling. The production speed considered in SLM process mainly involves in scanning speed and recoater speed (Kucukkoc, 2019). The capacity of a given AM machine is restricted by the two-dimensional area of the machine's building platform and its height. Thus, the parts in a batch are required to be placed into the building platform of the allocated machine in certain orientation without stack, and the maximal height can not exceed the machine height.

Based on the aforementioned characteristics, we denote this problem as the unrelated parallel additive manufacturing machine scheduling problem with orientation selection and 2D packing constraints. The objective is to minimize the makespan. To the best of our knowledge, our problem is a new practical variant of the classical batch processing machine (BPM) scheduling problem with several interesting features. First, because of the SLM process, the process time of one batch is not determined by the largest processing time of parts (Kucukkoc, 2019), but related to the total volume of parts and their support structures, as well as the maximum height of those parts. Second, most of BPM problems are modelled as one-dimensional knapsack problems. However, we consider a problem with two-dimensional rectangular packing and the parts sizes are not fixed but changing with the chosen orientation. Both batch machine scheduling problem and 2D packing problem are proven NP-hard, so is our problem.

To solve this problem, we first present a mixed integer linear programming (MILP) model and some strategies to strengthen the model. For larger scale instances, efficient heuristic algorithms are designed. The simulated annealing algorithmic framework is introduced, where eleven types of neighborhood operators are proposed to perform local search. For the packing subproblem, two fitting strategies are introduced based on the skyline representation for packing patterns, which are embedded to develop the random local search algorithm. In addition, data structure Trie is adopted not only to track the packing feasibility information of parts already examined, but also to control the effort spent in packing procedure. Furthermore, four greedy based post-optimization methods are provided to refine the solution. Finally, we evaluate the performance of the proposed algorithms with comprehensive analyses.

The main contributions of this paper are summarized as follows. First, this is the first work to consider orientation selection and two-dimensional packing simultaneously in the batch processing machine scheduling problem in AM. We propose the mathematical model and meta-heuristic algorithms for this problem. Second, in the packing process, practical factors related to orientations that influence the required processing time are considered, and new fitness evaluation strategies are designed. The experimental results show the proposed RandomLS packing algorithm performs better than the fixed-orientation scenarios. Third, data structure Trie is used to accelerate the packing procedure and post-optimization methods are designed to improve the solutions obtained from SA heuristic. Finally, extensive experiments are conducted to evaluate the proposed algorithms, and the effectiveness of considering orientation selection in machine scheduling is verified.

The remainder of this paper is organized as follows. Section 2 reviews the related works. The formal description and mathematical formulation is presented in Section 3, followed by the framework of the simulated annealing algorithm in Section 4. Section 5 is dedicated to the packing procedure, whereas post-optimization methods are explained in Section 6. The experiments are conducted to evaluate the performance of the proposed algorithms in Section 7. The final conclusion is drawn in Section 8.

2. Literature review

In this section, we review the literature related to the problem discussed in this work. We first summary the research on BPM and analyze papers for planning and scheduling of AM production (Section 2.1), and then introduce the common methods of how to select orientations from candidates (Section 2.2).

2.1. Planning and scheduling of AM production

BPM scheduling problems in different machine environments have been studied extensively. Based on the problem considered in this work, we mainly focus on the unrelated parallel BPMs scheduling problems with non-identical job sizes. Li et al. (2013) first proposed to schedule unrelated parallel BPMs with non-identical job sizes with the objective to minimize the makespan. Heuristics based on Best-Fit and longest processing time (LPT) rules are designed to solve the problem. Arroyo and Leung (2017b) introduced a variant with unequal release times and presented an MILP model. They also proposed several heuristics based on the first-fit and best-fit earliest job ready time rules. The same authors (Arroyo and Leung, 2017a) studied unrelated parallel batch machine scheduling with non-identical capacities and unequal ready times. An effective iterated greedy algorithm was provided to solve the problem. Later, Zhang et al. (2018) solved the unrelated parallel BPMs with non-identical job sizes and arbitrary release times through a random-keys genetic algorithm. A bi-objective batch processing problem with dual-resources on unrelated-parallel machines was addressed in Shahvari and Logendran (2017), where a bi-objective MILP model was developed in four layers and a Pareto optimal set was obtained.

From the planning and scheduling point of view in AM, the research is still limited. Li et al. (2017) introduced a production planning problem in additive manufacturing. An MILP model was presented with the objective of minimizing the average production cost per material volume. They used CPLEX to solve the model and developed two heuristics, best-fit and adapted best-fit heuristics. Chergui et al. (2018) addressed the production scheduling and nesting problem in AM with due date constraints. They proposed a heuristic approach to minimize the maximum lateness. Dvorak et al. (2018) studied an AM machine scheduling problem with a due date for each part, and its objective was to minimise the number of tardy parts. Fera et al. (2018) presented a single AM machine scheduling problem with the objective to minimize time consumption and material cost. Kucukkoc (2019) addressed the scheduling problem of multiple AM machines and developed several MILP models with the objective to minimize makespan. Besides, Zhang et al. (2020) considered the parallel batch processing machine scheduling in additive manufacturing with irregular packing. To the best of our knowledge, there is no research about the unrelated parallel BPMs scheduling problems with non-identical job sizes and two-dimensional packing in AM.

2.2. Orientation selection in AM

Since the orientation of each part has significant impact on the production cost and time, the selection of the build orientation for a part is considered to be an important research problem. The first selection criterion is the *support structure*. During the build process, support structure is used to hold the surface of part where the angle with respect to the horizontal axis is smaller than 45 degrees (Calignano, 2014). An example of the support structure is shown in Fig. 4. In general, support structures use the same building materials as the parts. That is, more support structure means more recoatering time. Thus, a common objective is to minimise the projected volume of support structures (Yang et al., 2003; Das et al., 2017). Another useful objective is to minimise the total area of surfaces which need the support structure (Canellidis et al., 2009). The second common selection criterion is the *surface roughness*. Surface roughness, caused by the staircase effect of layer-by-layer production, is mainly used to estimate the quality of parts. There are several objectives related to the roughness, including the average roughness (Pandey et al., 2004; Thrimurthulu et al., 2004), the weighted average roughness (Byun and Lee, 2006; Byun and Lee, 2006), and the total area of surfaces with estimated roughness above a certain limit (Ahn et al., 2007). Other criteria used to orientation determination are the *building time and cost* for one part (Lai and Chan, 1997; Alexander et al., 1998; Hur et al., 2001). Since there are different objectives, multi-attribute decision making methods are provided. The weighted sum of objectives are proposed in Byun and Lee (2006) and Canellidis et al., 2009. Meanwhile, some researchers choose to optimize the hierarchical objectives (Alexander et al., 1998). The pareto front approach is also studied recently (Wu et al., 2014; Padhye and Deb, 2011). Note that Zhang et al. (2017) is the only work considering the build orientation of multiple parts, but without considering two-dimensional packing.

3. Problem description and formulation

In this section, we provide a formal description and a mathematical model for the problem.

3.1. Problem description

Let D denote a set of heterogeneous AM machines. Each machine $m \in D$ is featured by setup time S_m, the scanning speed V_m, the recoater speed U_m, and size of its building platform, where (L_m, W_m, H_m) corresponds to its length, width, and height, respectively. Let I denote the parts to be produced. Each part $i \in I$ is featured by volume v_i and a set of predetermined orientation candidates N_i (Zhang et al., 2017). For each orientation $n \in N_i, (q_{in}, p_{in}, r_{in}, s_{in})$ are the corresponding length, width, height, and the volume of needed support structure. Each part is produced only once.

As shown in Section 1, all parts in I should be grouped into a set of batches K, which are further allocated to machines in D, while the

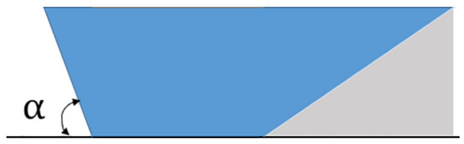

Fig. 4. The supported surface. If the angle α between the part surface and the horizontal axis is smaller than 45 degrees, support structure should be added. The gray area shows the added support structure.

parts must be successfully packed into the allocated machine without any overlap between each other. In this study, we consider the rectangular bounding box of each part and assume that the production area of each part is represented by the bounding box projected on the building platform during packing (Canellidis et al., 2013), and the packing of the parts in the same batch is considered as a 2D rectangular packing problem. We also assume that horizontal 90 degrees rotation of each part is allowed. During the packing procedure, each part is allowed to choose one of its orientation candidates. The processing time of a batch is a function of scanning time which is proportional to the total volume of parts and support structures, recoater time which is proportional to maximum height of the parts in this batch, and the setup time of the assigned machine (Kucukkoc, 2019). And the processing time of machine m is 0 if no parts are assigned to it. The maximum height of the parts in the batch can not exceed the height of the building platform of machine. The objective is to minimize the makespan of all machines.

In addition, we assume that the batch production process cannot be interrupted until it is finished. Once a batch is assigned to a machine, the batch is characterized by the features of the machine, including size, production time, etc. Therefore, in the following description of this paper, we will simply use the word "batch" without mentioning the assigned machine.

3.2. Mathematical formulation

This section presents an MILP model for the problem, which can be directly solved by commercial softwares. The parameters used in the mathematical model are defined as follows:

- S_m: the setup time of machine m;
- V_m: the scanning speed of machine m;
- U_m: the recoater speed of machine m;
- (L_m, W_m, H_m): the length, width, and height of the building platform of machine m;
- v_i: the volume of part i;
- $(q_{in}, p_{in}, r_{in}, s_{in})$: the corresponding length, width, height, and the volume of needed support structure when part i is in the orientation n;
- M: a sufficiently large positive number. The values of Big M for different constraints are specified in Section 3.2.3.

The variables used in the mathematical model are defined as follows:

- t: maximal completion time of all machines, namely, makespan;
- z_{km}: binary variable, equal to 1 if the k_{th} batch of machine m is used, and 0 otherwise;
- α_{ikm}: binary variable, equal to 1 if part i is assigned to the k_{th} batch of machine m, and 0 otherwise;
- b_{ij}: binary variable, equal to 1 if part i is at the left of part j, and 0 otherwise;
- c_{ij}: binary variable, equal to 1 if part i is in front of part j, and 0 otherwise;
- a_{in}: binary variable, equal to 1 if part i is placed in orientation n, and 0 otherwise;
- o_i: binary variable, equal to 1 if the width of the part i is parallel to the length of the machine, and 0 if part i is placed with its length parallel to the length of the machine;
- h_{km}: the height of the k_{th} batch of machine m, namely, the maximum height of parts in this batch;
- t_{km}: processing time of the k_{th} batch of machine m;
- (x_i, y_i): the coordinates of the front-left corner of part i.

Using these variables, and the notation given in the previous subsection, this problem can be formulated as follows.

$$\min\ t \tag{1}$$
$$\text{s.t. } t \geqslant \sum_{k\in K} t_{km}, \forall m \in D, \tag{2}$$
$$\sum\nolimits_{m\in D}\sum_{k\in K}\alpha_{ikm} = 1, \forall i \in I, \tag{3}$$
$$\sum_{i\in I}\alpha_{ikm} \leqslant Mz_{km}, \forall k \in K, m \in D, \tag{4}$$
$$t_{km} = S_m z_{km} + V_m\sum_{i\in I}(\alpha_{ikm}v_i + \alpha_{ikm}\sum_{n\in N_i}a_{in}s_{in}) + U_m h_{km}, \forall k \in K, m \in D, \tag{5}$$
$$x_i + \sum_{n\in N_i}a_{in}p_{in} \leqslant W_m + M(1-\alpha_{ikm}) + Mo_i, \qquad \forall i \in I, k \in K, m \in D, \tag{6}$$
$$x_i + \sum_{n\in N_i}a_{in}q_{in} \leqslant W_m + M(1-\alpha_{ikm}) + M(1-o_i), \ \forall i \in I, k \in K, m \in D, \tag{7}$$
$$y_i + \sum_{n\in N_i}a_{in}q_{in} \leqslant L_m + M(1-\alpha_{ikm}) + Mo_i, \qquad \forall i \in I, k \in K, m \in D, \tag{8}$$
$$y_i + \sum_{n\in N_i}a_{in}p_{in} \leqslant L_m + M(1-\alpha_{ikm}) + M(1-o_i), \ \forall i \in I, k \in K, m \in D, \tag{9}$$
$$x_i + \sum_{n\in N_i}a_{in}p_{in} \leqslant x_j + M(1-b_{ij}) + Mo_i, \forall i,j \in I, i \neq j, \tag{10}$$
$$x_i + \sum_{n\in N_i}a_{in}q_{in} \leqslant x_j + M(1-b_{ij}) + M(1-o_i), \forall i,j \in I, i \neq j, \tag{11}$$
$$y_i + \sum_{n\in N_i}a_{in}q_{in} \leqslant y_j + M(1-c_{ij}) + Mo_i, \forall i,j \in I, i \neq j, \tag{12}$$
$$y_i + \sum_{n\in N_i}a_{in}p_{in} \leqslant y_j + M(1-c_{ij}) + M(1-o_i), \forall i,j \in I, i \neq j, \tag{13}$$
$$b_{ij} + b_{ji} + c_{ij} + c_{ji} \geqslant \alpha_{ikm} + \alpha_{jkm} - 1, \forall i,j \in I, i < j, k \in K, m \in D, \tag{14}$$
$$\sum\nolimits_{n\in N_i}a_{in}r_{in} \leqslant h_{km} + (1-\alpha_{ikm})M, \forall i \in I, k \in K, m \in D, \tag{15}$$
$$h_{km} \leqslant H_m, \forall k \in K, m \in D, \tag{16}$$
$$\sum\nolimits_{n\in N_i}a_{in} = 1, \forall i \in I, \tag{17}$$
$$x_i, y_i \geqslant 0, \forall i \in I, \tag{18}$$
$$b_{ij}, c_{ij} \in \{0,1\}, \forall i,j \in I, \tag{19}$$
$$z_{km} \in \{0,1\}, \forall k \in K, m \in D, \tag{20}$$
$$\alpha_{ikm} \in \{0,1\}, \forall i \in I, k \in K, m \in D, \tag{21}$$
$$h_{km}, P_{km} \geqslant 0, \forall k \in K, m \in D, \tag{22}$$
$$o_i \in \{0,1\}, \forall i \in I, \tag{23}$$
$$a_{in} \in \{0,1\}, \forall i \in I, n \in N_i \tag{24}$$
$$t_{km} \geqslant 0, \forall k \in K, m \in D, \tag{25}$$
$$t \geqslant 0. \tag{26}$$

The objective (1) is to minimize the makespan which is the completion time of the last batch. Constraints (2) ensure the calculation of makespan. Constraints (3) guarantee that each part is processed only once by one machine. Constraints (4) ensure that a part cannot be assigned to an unused batch. Constraints (5) are used to calculate the processing time of one batch. The processing time of unused batches is automatically forced as 0.

Constraints (6)-(9) state that a part can not be placed outside the machine's platform no matter it is placed in horizontal or vertical direction. In constraints (6)-(9), if α_{ikm} equals to 1, it means that the part is placed on batch k of machine m, where L_m and W_m are used to restrict the placement. If α_{ikm} equals to 0, these constraints are relaxed. Constraints (6) and (7) are used to control the rotation. When α_{ikm} equals to 1, if o_i equals to 0, part i is placed with its length q_{in} parallel to the length L_m of machine m, thus constraints (6) are tight and constraints (7) are relaxed. When o_i equals to 1, the situation is reversed, where the width p_{in} is parallel to the length L_m of machine m. The same logic applies to constraints (8) and (9). Constraints (10)-(14) guarantee that if two parts are assigned in the same batch, they cannot overlap. Specifically, if $b_{ij} = 1$, constraints (10) and (11) force part i to be placed totally on the left of part j. Otherwise, they are relaxed because part i is not necessarily on the left of part j. The same logic applies to constraints (12) and (13). Constraints (14) guarantee that if two parts are assigned to the same batch, one of the four positioning relationship variables should be 1 and then overlap can be avoided. Constraints (15) are implemented to find the height of one batch and constraints (16) ensure the height of one batch does not exceed the height of the machine. Constraints (17) guarantee that only one orientation is chosen for every part. Constraints (18)-(26) are domain constraints for variables.

3.2.1. Linearization

Since constraints (5) are quadratic, the Big M and new variables e_{ikm} are introduced to do the linearization. The variable e_{ikm} represents the volume of the support structure of part i produced on the k_{th} batch of machine m. Then constraints (5) are replaced by constraints (27)-(29). Constraints (27) are used to calculate the processing time of one batch, which is the same as constraints (5). Constraints (28) guarantee that the volume of support structure used in calculation is consistent with the chosen orientation of the part. Constraints (29) are domain constraints for variables e_{ikm}.

$$t_{km} = S_m z_{km} + V_m \sum_{i \in I} (\alpha_{ikm} v_i + e_{ikm}) + U_m h_{km}, \forall k \in K, m \in D, \tag{27}$$

$$e_{ikm} \geqslant \sum_{n \in N_i} a_{in} s_{in} - (1 - \alpha_{ikm}) M, \forall i \in I, k \in K, m \in D, \tag{28}$$

$$e_{ikm} \geqslant 0, \forall i \in I, k \in K, m \in D. \tag{29}$$

3.2.2. Symmetry breaking constraints

There are two symmetry issues stemming from the identical batches of the same machine. Denote by k_m the number of batches preassigned to each machine m, and $\tilde{k}_m$ the number of batches actually used in each machine. First, there are $\binom{k_m}{\tilde{k}_m}$ possible options to choose the used batches. Second, the $\tilde{k}_m!$ permutations of used batches are identical for a machine. Thus, they result in $\prod_{m \in M} \left[\binom{k_m}{\tilde{k}_m} \tilde{k}_m! \right]$ equivalent solutions in total, which typically slows down the solving process.

To break the first type of symmetry, we can use symmetry breaking constraints (30) to allow batch k to be used only if batch $k-1$ is used first. With the help of constraints (30), the batches are utilised incrementally (starting from batch 1) on each machine. For second symmetry issue, constraints (31) breaks the symmetry of batches by ordering them according to their processing time.

$$z_{k-1,m} \geqslant z_{km}, \forall k \in K \setminus \{0\}, m \in D, \tag{30}$$

$$t_{k-1,m} \geqslant t_{km}, \forall k \in K \setminus \{0\}, m \in D. \tag{31}$$

3.2.3. Setting of Big M

In the mathematical model, Big M is used to turn on or turn off the enforcement of a constraint. The value of the Big M usually has a great impact on the computational time and thus needs to be chosen carefully. If M is too small, some feasible solutions may be cut off. On the other hand, the model may become numerically difficult. Therefore, we carefully define the value for every Big M in our model.

1) Constraints (4): the number of all parts, $M = |I|$.
2) Constraints (28): the largest volume of the support structure among all orientations of all parts, $M = \max_{i \in I} \max_{n \in N_i} s_{in}$.
3) Constraints (6) and (9): the largest width among all orientations of all parts, $M = \max_{i \in I} \max_{n \in N_i} p_{in}$.
4) Constraints (7) and (8): the largest length among all orientations of all parts, $M = \max_{i \in I} \max_{n \in N_i} q_{in}$.
5) Constraints (10): the largest width among all orientations of all parts plus the largest width among all machines, $M = \max_{i \in I} \max_{n \in N_i} p_{in} + \max_{m \in D} W_m$.
6) Constraints (11): the largest length among all orientations of all parts plus the largest width among all machines, $M = \max_{i \in I} \max_{n \in N_i} q_{in} + \max_{m \in D} W_m$.
7) Constraints(12): the largest length among all orientations of all parts plus the largest length among all machines, $M = \max_{i \in I} \max_{n \in N_i} q_{in} + \max_{m \in D} L_m$.
8) Constraints (13): the largest width among all orientations of all parts plus the largest length among all machines, $M = \max_{i \in I} \max_{n \in N_i} p_{in} + \max_{m \in D} L_m$.
9) Constraints (15): the largest height among all orientations of all parts, $M = \max_{i \in I} \max_{n \in N_i} r_{in}$.

4. The simulated annealing algorithmic framework

Simulated annealing (SA) is widely used to solve combinatorial optimization problems. It was motivated by the similarity between the metal annealing and the process of searching the optima in combinatorial optimization problems. It starts with an initial solution and continues to search among the neighboring space of the incumbent solution. To help escape from the local optima, SA allows to accept the worsen solutions with certain probability which is controlled by the temperature parameter. SA algorithm has been successfully introduced to solve the parallel machine scheduling problems (Lin and Ying, 2015; Xiao et al., 2015) and the integrated routing and two-dimensional loading problem (Wei et al., 2018). Therefore, SA is also adopted in this work. The pseudo-code of our method is shown in Algorithm 1.

Algorithm 1. SA

```
1: Construct the initial solution $S$; initialize the roulette wheel
2: $S_{best} \leftarrow S, T \leftarrow T_0, c \leftarrow 0$
3: while time limit is not reached and $T > T_{min}$ and $c < Count$ do
4:    for $i$ from 1 to $Len$ do
5:       Select a neighborhood operator $NS$ using roulette wheel
6:       Generate a feasible solution $S'$ from $S$ with $NS$
7:       if $cost(S') < cost(S)$ then
8:          $S \leftarrow S'$
9:          if $S'$ is better than $S_{best}$ then
10:            $S_{best} \leftarrow S'$
11:            $c \leftarrow -1$
12:            Update the roulette wheel
```

(continued on next page)

```
13:         end if
14:       else
15:         Set $S \leftarrow S'$ with the probability
            $p = exp(-\frac{cost(S')-cost(S)}{T})$
16:       end if
17:     end for
18:     $T \leftarrow T * \alpha$
19:     $c \leftarrow c + 1$
20: end while
21: return $S_{best}$
```

The $cost(S)$ is the makespan of the scheduled solution. Initially, a solution S is constructed (Line 1). In each iteration of the search process, the algorithm selects one neighborhood operator using roulette wheel and generates a new neighboring solution S' based on the incumbent solution S (Lines 5 and 6). The probability of one operator being chosen is dependent on the ratio of its fitness to the sum of the fitness values for all the operators. In this study, the fitness is set to be the number of times that the operator helps produce a new best solution. The packing algorithm is introduced to ensure the feasibility of S' during the generation process. If the new solution S' is better than S, it will be definitely accepted as the new incumbent solution. Otherwise, it is accepted with a probability of $exp(-\frac{cost(S')-cost(S)}{T})$, where T is the temperature parameter used to control the probability (Lines 7–16).

At the beginning, parameter T is set to T_0 (Line 2). Then, during the process, T decreases by multiplying the annealing rate α (Line 17) until it reaches to T_{min}. A large initial temperature T_0 means a big chance to accept worsen solutions, therefore helps escape the local optima. Along with the decrease of T, it tends to only accept better solutions. In addition, the algorithm is also terminated if the best solution is not improved for *Count* consecutive iterations. This avoids to spend a long time under low temperatures without improving the best solution. The value of *Count* is set as $\left\lfloor \sqrt{\frac{numberofparts}{numberofmachines}} \right\rfloor$ in this study. The construction of initial solution and design of neighborhood operators are explained in Sections 4.1 and 4.2, respectively.

4.1. Initial solution

Based on the observation that the setup time of a new batch contributes a lot to the objective function, we generate the initial solution using as fewer batches as possible, namely, each batch has to pack parts to the maximum of its capacity. In each iteration, a machine with the least makespan is chosen to create a new empty batch. Then, unpacked parts are sorted by decreasing heights, and chosen one by one until the total area just reaches the batch area. Here, the area of each part is calculated with its length multiplying the width from a randomly chosen orientation. Next, the packing algorithm described in Section 5 is invoked to check the feasibility. If no feasible packing pattern is found, remove the last part and pack again until obtaining a feasible packing pattern. After that, the list of unpacked parts is updated and the next iteration is repeated if the list is not empty.

4.2. Neighborhood operators

In this paper, 11 types of neighborhood operators are introduced with 3 basic operations: relocation, swap and split. Based on the current feasible solution, a neighborhood operator is chosen to form a new solution. Since the chosen neighborhood operator does not necessarily generate a feasible solution every time due to the packing failure, it is allowed to attempt for up to 20 times until a feasible solution is found.

4.2.1. Relocating operators

Five types of relocating operators are implemented to generate the neighboring solutions. The first type is the relocation of one batch, where one batch is randomly chosen and relocated to another machine. Fig. 5(a) shows an example, where batch 1 in machine 1 is relocated to machine 2. The second and third types are the relocation of one randomly chosen part to a different batch in the same machine and across machines respectively. To further enlarge the neighboring space, we also propose the other two types which are similar to the second and third types except that several parts are randomly chosen from a batch (Fig. 5(b) and (c)).

4.2.2. Swapping operators

Swapping operators exchange positions of batches or parts. Similarly to the relocating operators, five types are also developed, including swapping two batches from different machines, swapping two parts belonged two batches of the same machine or different machines, swapping several parts of two different batches in the same machine or across machines. The examples are shown in Fig. 6 (a-c).

4.2.3. Splitting operators

The splitting operator is to randomly choose a set of parts from one batch and use them to form a new batch. The new batch is then allocated to a randomly chosen machine. As shown in Fig. 7, parts 3 and 4 of batch 1 in machine 1 are moved to form a new batch 3 which is allocated to machine 2.

5. Heuristic for the packing subproblem

In this section, we solve the packing subproblem to check the feasibility of a given batch. First, the skyline representation of packing pattern is described. Next, two fitting strategies, *Best-Fit* and *First-Fit*, are proposed in terms of fitness evaluation for the placement of each part. Then, random local search heuristic algorithm is introduced to find the packing pattern for all parts. Finally, we present the acceleration strategy embedded in the procedure, data structure *Trie*.

5.1. Representation of a packing pattern—Skyline

During the packing procedure, we usually maintain several lists: the packed item list, the unpacked item list, and the remaining space list. The way to represent the remaining space affects the packing utility greatly. In this section, we introduce a widely-used representation of packing pattern—Skyline. Four corresponding fitness evaluations are also designed for the particular settings in our paper.

The skyline representation of a packing pattern was first proposed by Burke et al. (2004) and improved later by many researchers (Leung et al., 2011; Wei et al., 2017; Wei et al., 2018; Zhang et al., 2018). The priority scoring rules used in paper are adopted from the method in Wei et al. (2018).

As Fig. 8(a) shows, a skyline based space is the contour of a vertical segment, which is represented by five attributes (x, y, w, l_1, l_2), where (x, y) is the coordinates of the left endpoint of the segment, w is the width of the segment, l_1 and l_2 are the bound of two sides. The initial space is represented by a single line segment corresponding to the bottom of the machine platform. At each iteration, the selected part is placed adjacent to the specific segment line either at the left point or the right point. After the selected part is placed, the skyline is updated. A new segment corresponding to the top edge of the part is added, and other existing segments

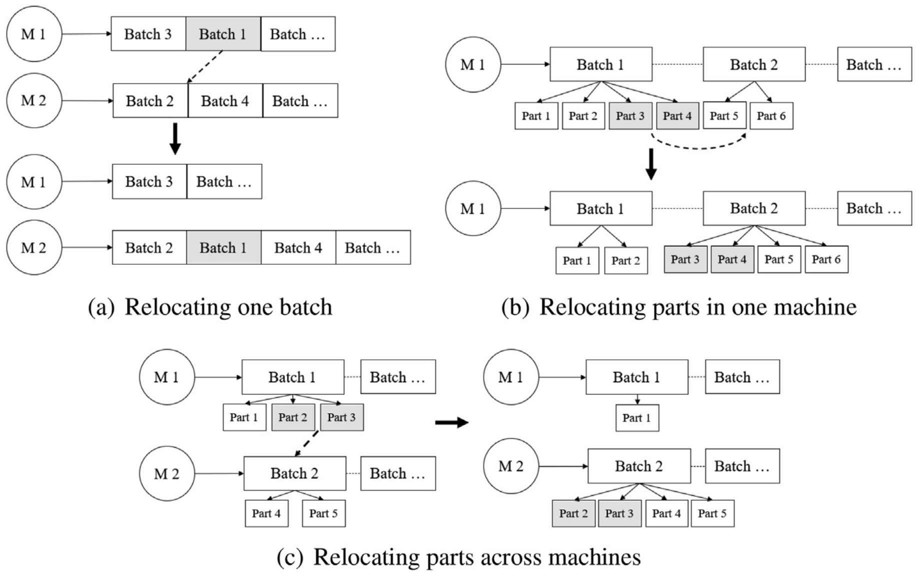

Fig. 5. Relocating operators.

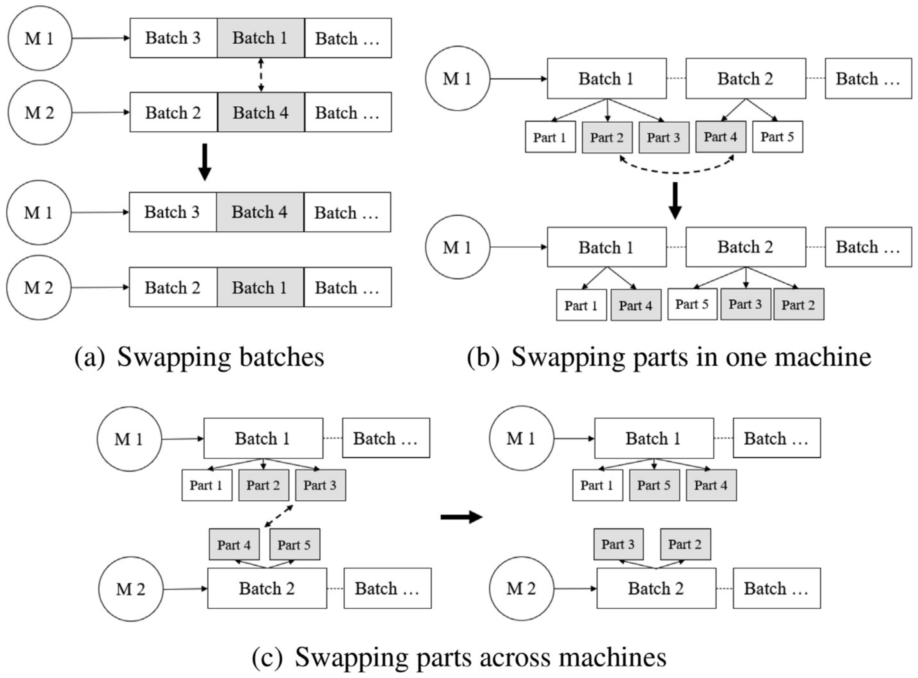

Fig. 6. Swapping operators.

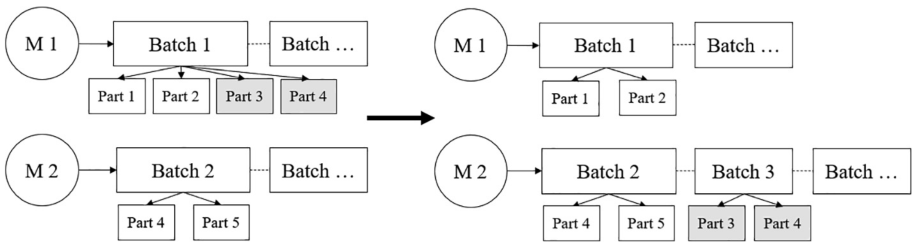

Fig. 7. Splitting operations.

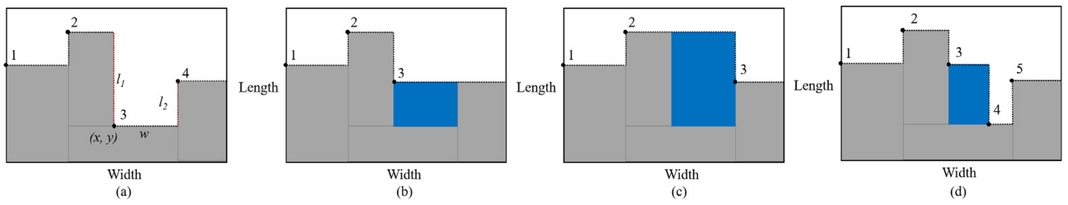

Fig. 8. Skyline representation and updating process.

affected are also updated. Fig. 8 (b)-(d) give three scenarios of updating the skylines after placing a part.

We design three criteria to evaluate the fitness when choosing a part to place in a specific space:

1) Priority evaluation. Fig. 9 shows ten scenarios corresponding to different priority numbers, where higher priority number represents better fitness. Especially three scenarios are included in case (h). During the evaluation, we calculate the priority number for all orientations of each part. When two or more situations have the same priority number, we randomly choose one.
2) Priority-Height evaluation. It is the same as the priority evaluation except that lower height is chosen to break ties for the same priority number.
3) Priority-Support evaluation. It is the same as the priority evaluation except that less volume of support structure is used to break ties.

The motivation of the last two fitness evaluations comes from the influence of parts' orientations to the time consumption. Therefore, the height or support structure of each part are added into fitness evaluation to lower the maximal height or the volume of the batch, thus, reduce process time.

5.2. Packing strategies

Given a sequence of parts in a batch, we implement two strategies *First-Fit* and *Best-Fit* to determine the position of the parts. The detailed procedures are shown in Algorithm 2 and 3 respectively. The *First-Fit* strategy places the parts one by one according to the given sequence by identifying the first feasible space, while the *Best-Fit* strategy always tries to fill the bottom left space with the best suitable part. More specifically, the *First-Fit* strategy checks every remaining space for each part until the first available space is found no matter in which orientation. Once found, we place the part in the space and update the space list, then move to the next unpacked part. The process is repeated until all parts are placed. Otherwise, the packing fails. However, in each iteration of the *Best-Fit* strategy, the bottom left space is first selected. Then, for each unpacked part, all orientations are evaluated and the one with the best fitness is kept. Next, the part with the highest fitness is chosen to place and the space list is updated. The procedure is repeated until all the parts are placed in or no spaces are available. The function *Fit* in Algorithm 3 indicates one of the fitness evaluation introduced in Section 5.1. Compared to the *First-Fit* strategy, the *Best-Fit* strategy is apparently more time-consuming since it evaluates all possible packing patterns for a specific space in each iteration. However, the *Best-Fit* strategy is more likely to find a better packing pattern in terms of the objective function. In the experiments, we compare the performance of these two strategies.

Algorithm 2. FirstFitPacking

```
Require: batch $b$, part list $P$
1: Initial the space $s$ with the length and width of the batch;
2: $S = \{s\}, num \leftarrow 0$
3: for each part $i \in P$ do
4:    for each orientation $n \in N_i$ do
5:      for each space $s \in S$ do
6:        if part $i$ in orientation $n$ can be placed in space $s$ then
7:          PlacePart($i, n, s$);
8:          Update space list $S$;
9:          $num \leftarrow num + 1$;
10:         Goto Out;
11:       end if
12:     end for
13:   end for
14: end for
15: Out:
16: if $num < |P|$ then
17:   return infeasible
18: else
19:   return best found packing pattern
20: end if
```

Algorithm 3. BestFitPacking

```
Require: batch $b$, part list $P$
1: Initial the space $s$ with the length and width of the batch;
2: $S = \{s\}, num \leftarrow 0$
3: Whlile $num < |P|$ and $S! = NULL$ do
4:   Find the bottom left space in $S$;
5:   for each unpacked part $i \in P$ do
6:     for each orientation $n \in N_i$ do
7:       $fitness \leftarrow Fit(i^*, n^*, s^*)$;
8:       if part $i$ in orientation $n$ can be placed in space $s$ and
     fitness is better than fitness* then
9:         $i^* \leftarrow i, n^* \leftarrow n, fitness^* \leftarrow fitness$;
10:      end if
11:    end for
12:  end for
13:  PlacePart($i^*, n^*, s^*$);
14:  Update space list $S$;
15:  $num \leftarrow num + 1$;
16: end while
17: if $num < |P|$
18:   return infeasible
19: else
20:   return best found packing pattern
21: end if
```

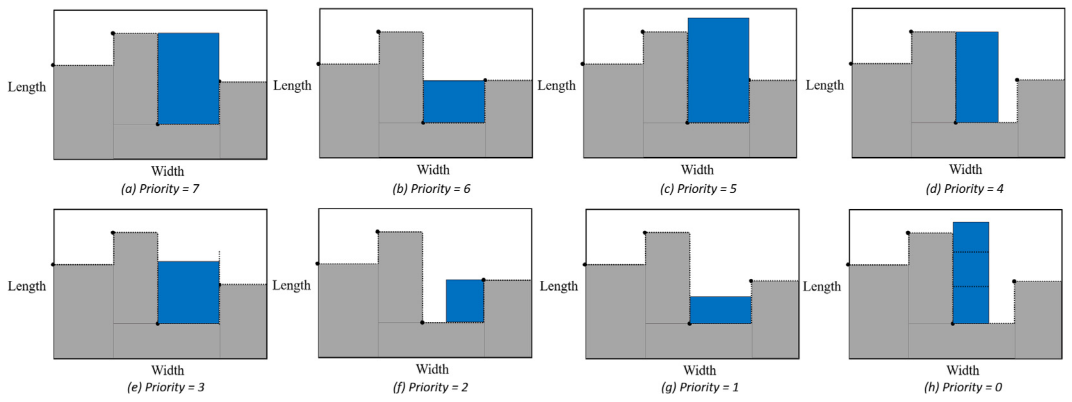

Fig. 9. Priority setting for different packing patterns.

5.3. Random local search heuristic for the packing subproblem

This Random Local Search (RandomLS) algorithm takes the sequence of parts in a batch as input, and output whether they can be placed in the limited machine area without overlap. If feasible packing patterns are found, the packing pattern with the least processing time is chosen and recorded; otherwise the batch can not be allocated to the machine. To accelerate the whole process, the packing result of the batch is stored in a date structure Trie, which is introduced in Section 5.4.

The detailed procedure is shown in Algorithm 4. Given a batch *b*, a set of parts *P*, and parameter *iter*, the parts are sorted to form a sequence according to designed sorting rules at first, and then *FitPacking* is called to get the packing pattern. *FitPacking* indicates any one of the packing strategies introduced in Section 5.2 (Lines 2–3). If no feasible packing pattern is found, two randomly selected parts are swapped and *FitPacking* is called iteratively until a feasible packing pattern is returned (Lines 4–7). Parameter *iter* here is set to be twice of the length of part list to limit the swap operations for each call of RandomLS. If the processing time of the new packing pattern v' is less, the best v is updated (Lines 8–9).

Algorithm 4. RandomLS

```
Require: $b, P, iter$
1: for each sorting rule do
2:   Sort P using the sorting rule
3:   $v_{best} \leftarrow$ FitPacking($b, P$),$i = 0, iter = 2 * |P|$
4:   while no packing pattern is found and $i < iter$ do
5:     Generate a new sequence $P'$ by swapping two randomly selected parts in $P$
6:     $v' \leftarrow$ FitPacking($b, P'$), $i \leftarrow i + 1$
7:   end while
8:   if $v' < v_{best}$ then
9:     $v_{best} = v'$
10:  end if
11: end for
12: ifno packing pattern is found then
13:   return infeasible
14: else
15:   returnbest found packing pattern
16: end if
```

The following sorting rules are used in our implementation:

- Sort the parts by height in decreasing order;
- Sort the parts by area in decreasing order;
- Sort the parts by length in decreasing order;
- Sort the parts by width in decreasing order;
- Sort the parts by support structure in decreasing order.

Take sorting by height for example, we set the height of each part as the minimum height among all its orientations, and then sort the parts by the height in decreasing order. The same way is used for the rest sorting rules.

5.4. Acceleration strategy

In order to speed up the procedure, data structure Trie is introduced not only to track the feasibility information of parts in the corresponding batch already examined, but also to control the effort spent on the packing part. This strategy is adapted from Wei et al. (2014); Wei et al., 2018, which was used to track the loading feasibility in vehicle routing problem. Due to the heterogeneous machines in our study, a particular trie is built for each machine. For a given batch, before invoking the RandomLS packing heuristic, we first retrieve the feasibility information from the corresponding Trie, and then different actions are taken according to the returned information. The stored information is a tuple $(packed, num, packingpattern)$, where *packed* is the feasibility information and *num* is number of calls of RandomLS on this batch. Five cases are classified based on the value of this tuple:

1. $packed == NULL$ (the batch does not exist in the Trie): The RandomLS is called to find the packing pattern, and the result is stored into Trie;
2. $packed == feasible, num < maxNum1$(feasible packing pattern of the batch exists in the Trie, and the batch has been retrieved fewer than $maxNum1$ times): The RandomLS is called to find the packing pattern, and the result is stored into Trie;
3. $packed == feasible, num \geqslant maxNum1$(feasible packing pattern of the batch exists in the Trie, and the batch has been retrieved no less than $maxNum1$ times): The packing pattern is implemented immediately;
4. $packed == infeasible, num < maxNum2$ (the batch has been packed but failed to find a feasible packing pattern, and the batch has been retrieved less than $maxNum2$ times): RandomLS is called to find the packing pattern, and the result is stored into Trie;
5. $packed == infeasible, num \geqslant maxNum2$ (the batch has been packed but failed to find a feasible packing pattern, and the batch has been retrieved no less than $maxNum2$ times): Return failure.

The parameters *maxNum*1 and *maxNum*2 are used to restrict the maximum allowed calls. The motivation of *maxNum*1 is to expand the diversity of feasible packing patterns of the same batch. In the experiment, *maxNum*1 and *maxNum*2 are set to be 2 and the number of parts in the batch, respectively. By setting this limitation, not too much effort is spent on the same batch, which can improve the efficiency of our approach. In the experiments, 15$\%$ calls of Trie successfully return *feasible* or *infeasible*, we can see that the utilization rate of data structure is very high.

6. Post-optimization

To further improve the solution obtained from the SA heuristic, we propose four post-optimization methods, which are all greedy strategies.

1. Makespan based greedy method: move one randomly chosen batch from the machine with the largest makespan to the one with the shortest makespan. The aim is to balance the workload among all machines.
2. Util based greedy method: move a randomly chosen part from the batch with the largest utilization rate to the batch with the smallest utilization rate. The utilization rate indicates the ratio of packed area to the whole batch area.
3. Height variance based greedy method: move a randomly chosen part from the batch with the largest height variance to the batch with the smallest height variance. The height variance of a batch is calculated by the heights of all the parts packed on.
4. Height based greedy method: change the orientation of the highest part in each batch and pack again. The aim is to reduce the maximal height of the batch, where the highest part determines the height of each batch, which affects the recoater time.

The stop criteria for the first three methods are that the number of consecutive non-improved iterations exceeds 20, and these four methods are performed successively. The factors that influence the makespan involve various aspects including parts assignment, part properties and packing pattern, and all these factors are correlative. For example, changing the orientation of a part may reduce the height of a batch but increase the support volume simultaneously. Therefore, we cannot guarantee the validity of the greedy methods. Actually, the first three greedy algorithms yield only tiny improvements for a few instances in the experiments. However, because they are extremely fast, we keep them included in the final design of our algorithms.

7. Experiments and analysis

To evaluate the performance of the proposed algorithms, we conduct several computational experiments on newly generated instances (Section 7.1). First, parameter tuning is conducted for SA in Section 7.2. Second, the results obtained by solving the mathematical model and the heuristic algorithms are compared in Section 7.3. We also demonstrate the advantage of considering orientation selection by comparing with fixed-orientation cases and shows the comparison of different packing algorithms. Finally, analyses on the utility of operators, greedy search methods and data structure Trie are given in Section 7.4. All the heuristic algorithms are implemented in Java and the experiments are conducted on an Intel(R) Xeon(R) Silver 4216 clocked at 2.10 GHz with 502 GB RAM. The maximal computational time for each instance is set as 600 CPU seconds. The MILP model is solved with ILOG CPLEX 12.8, and computing time for each instance is limited to 7200 CPU seconds. The test instances and detailed solutions can be downloaded at http://www.computational-logistics.org/orlib/2L-BPM.

7.1. Data generation

Since no exactly the same studies have been found yet, we generate new instances from 3D hardware designs in practical use. 88 parts are downloaded in form of "stl" file from the website "*Thingiverse*", which is dedicated to share primarily free, open source hardware designs. For each part, information of several orientations is collected, including width, length and volume of support, using a 3D printing application named *Repetier-Host*. Three types of additive manufacturing machines are generated based on Kucukkoc's paper (Kucukkoc, 2019).

To diversify the datasets, seven classes of instances are generated, and each class has 20 instances. For each class, the total numbers of machines and parts are specified, and their detailed types are randomly chosen from the aforementioned types. The detailed parameters are shown in Table 1.

7.2. Parameter tuning

To fulfill the potency of SA, it is crucial to adjust values of multiple parameters such as initial temperature T_0, cooling rate α, and the length of Markov Chain *Len*. The length *Len* is set to be proportional to the number parts, namely $Len = numberofmachines * \ numberofparts * \ \beta + 1000$. Because the numbers of machines and parts in small instances are small, 1000 extra steps are added to the *Len* to exploit the local search fully. Besides, the minimum temperature is simply set as 0.1.

We generate another four new instances for each class to conduct this parameter tuning. Since there is no obvious correlation between any pair of parameters, we analyze the possible values of one parameter each time by fixing the other parameters (Kim et al., 2002). The average *Makespan* used as indicator for performance analyses. Since we propose two packing strategies and three fitness evaluations in Section 5, four methods are generated in total. The parameter space definition and the best parameters of these methods are summarized in Table 2. Note that the notation in Table 2 is also used in the following sections.

7.3. Results comparison

7.3.1. Comparison with different solving methods

To evaluate the performance of the proposed SA algorithm framework, we compare the best heuristic results with solutions obtained by solving MILP model given in Section 3 with CPLEX solver. Within 7200 CPU seconds, class 1 is the only class solved to optima by CPLEX, except one instance. For larger instances in class 2, feasible solutions are also recorded in Table 3 for comparison with the proposed heuristic algorithm, where *Makespan* indicates the makespan of each instance and T_{run} is the running time.

For class 1, although the SA algorithm doesn't solves the instances to optima, the average gap between heuristic results and the optimal solutions is less than 1.8$\%$ which is very small, and the SA algorithm only uses one-eighth time of CPLEX. This also demonstrates the difficulty of this problem. For class 2, no instances are solved to optima by CPLEX within 2 h, but the SA algorithm achieves better results on 18 among 20 instances with much less time. The average improvement of solutions is up to 19.6$\%$. Therefore, the proposed SA algorithm is competitive in solution quality and computing speed, which is valuable in practical use.

Meanwhile, the detailed results obtained by four heuristic methods on the 7 classes are reported in Table 4, where *Makespan* indicates average makespan (in units of hour) of 7 classes, *batch* is the average number of used batches, T_{best} represents the average time (in units of second) when the best solution is found, and

Table 1
Parameters for data generation.

Class	1	2	3	4	5	6	7
# machines	2	2	3	4	5	6	7
# parts	20	50	100	200	300	400	500

Table 2
SA parameter tuning results for four methods.

Parameters	Range	FF	BF_P	BF_PS	BF_PH
T_0	[5,15]	5	5	15	5
α	[0.6,0.95]	0.9	0.7	0.7	0.7
β	[0.5,2]	1.5	2	1	2

* FF: *FirstFit* packing strategy.
* BF: *BestFit* packing strategy.
* P: Priority evaluation.
* PS: Priority-Support evaluation.
* PH: Priority-Height evaluation.

Table 3
Comparison with CPLEX and SA.

	*Inst.	Cplex			SA			Inst.	Cplex			SA	
		Optimility	*Makespan*	T_{run}	*Makespan*	T_{run}			Optimility	*Makespan*	T_{run}	*Makespan*	T_{run}
Class 1	1	Optim.	20.19	29	20.31	2	Class 2	1	Feas.	78.19	7259	79.18	636
	2	Optim.	20.76	30	21.20	1		2	Feas.	108.45	7202	74.24	266
	3	Optim.	29.62	25	31.24	4		3	Feas.	103.00	7201	89.21	97
	4	Optim.	41.51	54	41.92	3		4	Feas.	76.03	7202	75.51	5
	5	Optim.	25.35	25	25.74	4		5	Feas.	89.60	7201	86.57	350
	6	Optim.	29.89	8	29.93	1		6	Feas.	136.34	7201	64.92	366
	7	Optim.	34.40	63	34.76	0		7	Feas.	67.53	7201	62.79	95
	8	Optim.	35.98	32	36.20	1		8	Feas.	115.01	7201	80.83	542
	9	Optim.	29.27	15	29.63	2		9	Feas.	84.04	7201	84.93	476
	10	Optim.	40.98	25	42.66	4		10	Feas.	127.56	7201	83.88	245
	11	Optim.	40.03	34	40.65	0		11	Feas.	102.81	7201	80.56	236
	12	Optim.	37.25	70	38.16	1		12	Feas.	70.78	7210	68.15	8
	13	Optim.	31.46	439	31.71	5		13	Feas.	93.06	7214	82.69	118
	14	Optim.	25.74	43	26.02	1		14	Feas.	87.88	7201	77.42	338
	15	Optim.	31.45	30	31.70	1		15	Feas.	59.56	7213	58.41	68
	16	Feas.	31.89	7200	32.81	10		16	Feas.	61.52	7201	58.30	31
	17	Optim.	29.92	23	31.80	2		17	Feas.	85.88	7201	81.25	43
	18	Optim.	32.38	50	32.63	2		18	Feas.	64.57	7205	57.45	33
	19	Optim.	30.44	44	30.57	3		19	Feas.	80.62	7201	77.23	341
	20	Optim.	31.82	9	31.96	2		20	Feas.	92.42	7201	68.76	94
Avg.			31.52	55.16	**32.08**	**2.45**				89.24	7205.90	**74.61**	**219.40**

* Note that instance 16 is excluded when computing the average T_{run} for CPLEX of class 1.

T_{run} gives the average running time (in units of second). It shows that BF_PH performs best on all 7 classes in terms of *Makespan*, although BF_PH does not use the least number of batches. For the number of used batches, BF_PS performs best on classes 1, while BF_P performs better on the rest classes. Therefore, the number of batches does not necessarily decide the final makespan. Moreover, for classes 1–5, BF_P requires the least number of batches on almost all classes, but does not achieve the best makespan among four methods possibly due to neglecting the impact of parts' height on recoater time and support volume on scanning time. Table 5 shows the number of best results obtained by each method on 20 instances in each class. BF_PH shows absolute superiority on all classes. In addition, BF_PS, which considers support structure while packing, does not show obvious advantages. Compared with three methods based on *Best-Fit*, FF cannot outperform on any indicator.

From the overall performance of the four methods, we can conclude that (1) *Best-Fit* packing strategy is more effective than *First-Fit* packing strategy; (2) Packing patterns of each batch are more important compared to the number of used batches; (3) The fact that BF_FH performs the best indicates that height is an important factor considered in packing process.

7.3.2. Comparison with fixed-orientation

To assess the advantage of taking orientations selection of parts into consideration in the packing process, we compare the results with and without orientation selection. In the case of fixed orientation, one orientation is determined for each part in certain way, and parts are enforced to be packed in the fixed orientation but rotation is still allowed. Here, four ways are proposed to fix the orientation based on the common sorting rules used in other similar studies. The detailed rules are shown in below:

- Fix Height (FH): select the orientation with the minimum height for each part, and sort the parts in descending height;
- Fix Support (FS): select the orientation with the minimum support structure for each part, and sort the parts in descending support structure;
- Fix Area (FA): select the orientation with the minimum area structure for each part, and sort the parts in descending area;

Table 4
Result comparison for four methods.

Class	BF_PH			BF_PS			BF_P			FF		
	Makespan	*batch*	T_{best}	*Makespan*	*batch*	T_{best}	*Makespan*	*batch*	T_{best}	*Makespan*	*batch*	T_{best}
1	**32.08**	2.10	2.45	32.93	**2.00**	2.45	33.56	2.05	**0.25**	37.58	3.10	3.40
2	**74.61**	2.80	219.40	74.88	2.80	168.80	76.23	**2.25**	**17.35**	83.73	6.10	35.65
3	**99.62**	5.60	247.15	100.22	4.40	323.75	101.00	**3.60**	90.10	108.78	11.35	**58.60**
4	**148.16**	12.10	365.85	148.68	9.35	437.65	148.89	**6.70**	404.95	161.18	22.35	**125.70**
5	**184.24**	19.55	397.00	185.56	15.40	501.30	185.03	**11.30**	474.40	199.27	33.75	**264.65**
6	**200.89**	21.35	592.85	202.00	16.90	653.40	202.27	**12.15**	571.70	217.37	43.55	**446.45**
7	**225.15**	31.75	**536.30**	227.27	24.65	729.25	225.26	**17.75**	638.45	245.30	57.00	592.70
Avg.	**137.82**	13.61	337.29	138.79	10.79	402.37	138.89	**7.97**	313.89	150.46	25.31	**218.16**

Table 5
Count of best results for 4 methods.

Class	1	2	3	4	5	6	7	Total
BF_PH	**13**	**11**	**12**	**9**	**12**	**11**	**9**	**77**
BF_PS	7	6	4	7	4	5	4	37
BF_P	0	3	4	4	4	4	7	26
FF	0	0	0	0	0	0	0	0

Table 6
Result comparison in fixed orientations.

	Makespan	batch	T_{best}	T_{run}	max_height	support
MO	**137.82**	13.61	**337.29**	**310.55**	84.84	192235.43
FH	138.34	14.90	383.14	385.81	**61.99**	**181138.08**
FA	144.97	**8.22**	461.00	511.14	258.16	389287.97
FS	138.80	11.97	486.32	538.39	169.88	185765.92
RD	142.61	11.19	487.85	546.66	166.67	257015.43

* MO: BF_PH with multiple orientations.

- Random (RD): randomly select an orientation for each part, and sort the parts randomly.

Since BF_PH performs the best among four methods, the comparative experiments are executed using BF_PH method. Table 6 reports the detailed comparison results obtained by BF_PH using these four fixing rules and orientation selection. We can see that our method shows advantages on *Makespan*, T_{best} and T_{run}. Although the gap of makespan between FH and our method is small, our method requires much less running time and smaller number of batches. In terms of any indicator, FS, RD and FA are not competitive. Note that FA uses the least number of batches but performs worst on makespan. Table 7 shows the number of the best results obtained by these rules on 20 instances in each class. Our method performs better on more instances. Under comparison, FH and FS are inferior. The reason is that FH only focuses on minimising the height, and FS only focuses on minimising the support structure. Pursuing single objective like height or support structure leads to the neglect of another objective. Considering different orientations, we can reach more search space and balance different objectives to find better solutions (see Table 8).

7.3.3. Comparison with different packing algorithms

In our algorithm, RandomLS heuristic is used for packing of batches. To evaluate the performance of RandomLS, another two classical heuristics simulating annealing (SA) and tabu search (TS) are implemented for comparison. The comparison results in Table 7.3.3 show our algorithm performs better with less time when using RandomLS. Although RandomLS is relatively simpler than SA and TS, the packing problem is the subproblem of the whole machine scheduling problem, and it takes account in both packing feasibility and batch processing time minimization.

7.4. Result analysis

7.4.1. Usage of operators

We analyse the usage of the 11 operators. Fig. 10 shows the number of accepting a new solution when the operator is applied, and the ratio of accepting a new solution to the number of usage is labeled above each bar. We can see that the operator OP_11 is used the most often, following by OP_3 and OP_8, the accept ratio of operator OP_2 is highest. The accept ratio is 45% on average.

Table 7
Count of best results for all fixed-orientation situation.

Class	1	2	3	4	5	6	7	Total
MO	**11**	**11**	**9**	6	**11**	**8**	**8**	**64**
FH	8	5	5	3	4	6	6	37
FA	0	0	0	0	0	0	0	0
FS	4	4	5	**11**	4	6	6	40
RD	0	0	1	0	1	0	0	2

Table 8
Result comparison with different packing algorithm.

Class	RandomLS			SA			TS		
	Makespan	*batch*	T_{best}	*Makespan*	*batch*	T_{best}	*Makespan*	*batch*	T_{best}
1	32.08	2.10	2.45	32.13	2.15	19.40	32.09	2.05	17.65
2	74.61	2.80	219.40	74.60	3.05	313.30	74.76	3.05	355.10
3	99.62	5.60	247.15	99.76	5.90	342.25	99.38	5.90	412.30
4	148.16	12.10	365.85	148.93	11.35	508.20	148.89	12.20	474.25
5	184.24	19.55	397.00	184.81	18.75	627.60	186.19	18.10	564.55
6	200.89	21.35	592.85	201.18	22.25	770.20	202.55	24.80	620.70
7	225.15	31.75	536.30	226.69	32.95	728.85	226.26	30.50	738.40
Avg.	**137.82**	**13.61**	**337.29**	138.30	13.77	472.83	138.59	13.80	454.71

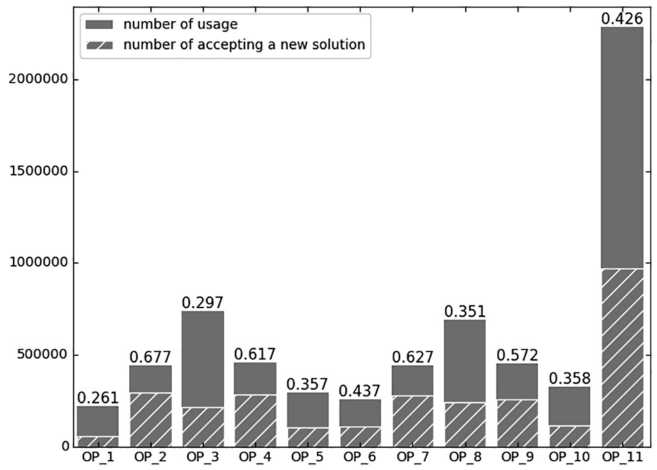

Fig. 10. Accept ratio of 11 operators. OP_1: relocate one batch; OP_2: relocate one part in one machine; OP_3: relocate one part across machines; OP_4: relocate parts in one machine; OP_5: relocate parts across machines; OP_6: swap batches; OP_7: swap two parts in one machine; OP_8: swap two parts across machines; OP_9: swap several parts in one machine; OP_10: swap several parts across machines; OP_11: splitting.

7.4.2. Performance of post-optimization methods

Table 9 shows the total number of improved solutions with the four post-optimization strategies for BF_PH method. The effectiveness of height based greedy method is the most obvious. Results for more than 16.4% instances are improved during Post-optimization and their performance are elevated by 1.6%.

7.4.3. Utility of Trie

Table 10 shows the utility of data structure Trie for BF_PH on each class. 22.9% of overall packing calls retrieve either feasible or infeasible answers from Trie. On smaller instances in class 1, it is easier to find answers from Trie, because smaller number of parts corresponds to fewer combinations of parts, therefore, fewer packing patterns. Overall, 22.4% of the packing calls can find answers from Trie, which helps save plenty of searching time to some extent. In addition, see from the value of TSU/TS, up to 73% of the feasible results retrieved from Trie are put into use instantly on average.

8. Conclusions

In this paper, we solved an unrelated parallel additive manufacturing machine scheduling problem with orientation selection and 2D packing constraints. In accordance with the characteristics of the problem, we present an MILP model for the problem. Then, efficient SA algorithmic framework is developed to solve this highly constrained problem. For the packing subproblem, we introduce the random local search algorithm with two fitting strategies based on the skyline representation of packing pattern. Moreover, data

Table 9
Utility of greedy search for BF_PH.

Class	1	2	3	4	5	6	7	Total
GreedyMakespan	0	0	0	0	0	0	4	5
GreedyHeight	0	0	2	0	5	7	4	**18**
GreedyUtil	0	0	0	0	0	0	0	0
GreedyHeightIndicator	0	0	0	0	0	0	0	0

Table 10
Utility of Trie for BF_PH.

Class	PN	TS	TSU	TSU/TS	TF	TS/Total	TSU/Total	TF/Total	(TS + TF)/Total
1	375474	426619	344901	0.81	24379	0.516	0.417	0.029	0.546
2	1194393	249735	156795	0.63	37191	0.169	0.106	0.025	0.194
3	1008957	200870	112726	0.56	4698	0.165	0.093	0.004	0.169
4	1464136	234158	150883	0.64	1022	0.138	0.089	0.001	0.138
5	1958008	457169	315579	0.69	1352	0.189	0.131	0.001	0.190
6	2074890	568788	422663	0.74	735	0.215	0.160	0.000	0.215
7	2335260	886207	694141	0.78	238	0.275	0.215	0.000	0.275
Tot.	10411118	3023546	2197688	0.73	69615	0.224	0.163	0.005	0.229

* *PN*: times of not retrieving any result from Trie and resorting to packing algorithm.
* *TS*: times of successfully retrieving feasible packing pattern from Trie
* *TSU*:times of successfully retrieving feasible packing pattern from Trie and put into use instantly.
* TSU/TS: TrieSuccessUse/ TrieSuccess.
* *TF*: times of successfully retrieve infeasible packing pattern from Trie.
* $Total = TS + TF + PN$.

structure Trie is introduced to accelerate the packing procedure by not only recording the packing information but also control the effort spent by packing procedure. Furthermore, four greedy based post-optimization methods are designed to refine the solution. Finally, extensive numerical experiments are conducted and show that the SA algorithm with BF-PH performs the best. The effectiveness of orientation selection is demonstrated to help improve the makespan in AM. Besides, some interesting observations are provided.

Although we have considered many constraints in this work, our model developed here can still be extended in several ways. This study is based on the SLM process and parts cannot be put above other parts. However, on some other AM process, it allows parts to be stacked on each other, which imposes three dimensional constraints on this problem. Moreover, the due dates constraints and the objective of minimizing total lateness can be considered and integrated into the model. We hope our study can give researchers an insight into the machine scheduling problems in AM.

CRediT authorship contribution statement

Yuxin Che: Conceptualization, Software, Validation, Formal analysis, Methodology, Investigation, Data curation, Writing - original draft, Writing - review & editing. **Kanxin Hu:** Conceptualization, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing. **Zhenzhen Zhang:** Investigation, Writing - original draft. **Andrew Lim:** Supervision, Project administration.

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.

Acknowledgements

The authors thank the two anonymous reviewers whose comments helped to improve and clarify this manuscript. This research was partially supported by the National Nature Science Foundation of China (Grant No. 71871070), NRF Singapore [Grant NRFRSS2016-004], and MOE Academic Research Fund of Singapore Tier 2 [Grant MOE2017-T2-2–153].

References

Ahn, D., Kim, H., Lee, S., 2007. Fabrication direction optimization to minimize post-machining in layered manufacturing. Int. J. Mach. Tools Manuf 47, 593–606.

Alexander, P., Allen, S., Dutta, D., 1998. Part orientation and build cost determination in layered manufacturing. Comput. Aided Des. 30, 343–356.

Arroyo, J.E.C., Leung, J.Y.-T., 2017a. An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non-identical capacities and unequal ready times. Comput. Ind. Eng. 105, 84–100.

Arroyo, J.E.C., Leung, J.Y.-T., 2017b. Scheduling unrelated parallel batch processing machines with non-identical job sizes and unequal ready times. Comput. Oper. Res. 78, 117–128.

Burke, E.K., Kendall, G., Whitwell, G., 2004. A new placement heuristic for the orthogonal stock-cutting problem. Oper. Res. 52, 655–671.

Byun, H.S., Lee, K.H., 2006. Determination of optimal build direction in rapid prototyping with variable slicing. Int. J. Adv. Manuf. Technol. 28, 307–313.

Byun, H.-S., Lee, K.H., 2006. Determination of the optimal build direction for different rapid prototyping processes using multi-criterion decision making. Robot. Comput.-Integr. Manuf. 22, 69–80.

Calignano, F., 2014. Design optimization of supports for overhanging structures in aluminum and titanium alloys by selective laser melting. Mater. Design 64, 203–213.

Canellidis, V., Giannatsis, J., Dedoussis, V., 2009. Genetic-algorithm-based multi-objective optimization of the build orientation in stereolithography. Int. J. Adv. Manuf. Technol. 45, 714–730.

Canellidis, V., Giannatsis, J., Dedoussis, V., 2013. Efficient parts nesting schemes for improving stereolithography utilization. Comput. Aided Des. 45, 875–886.

Chergui, A., Hadj-Hamou, K., Vignat, F., 2018. Production scheduling and nesting in additive manufacturing. Comput. Ind. Eng. 126, 292–301.

Das, P., Mhapsekar, K., Chowdhury, S., Samant, R., Anand, S., 2017. Selection of build orientation for optimal support structures and minimum part errors in additive manufacturing. Comput.-Aided Design Appl. 14, 1–13.

Dvorak, F., Micali, M., Mathieug, M., 2018. Planning and scheduling in additive manufacturing. Inteligencia Artif. 21, 40–52.

Fera, M., Macchiaroli, R., Fruggiero, F., Lambiase, A., 2018. Production management fundamentals for additive manufacturing. 3D Printing, 71–88..

Guo, N., Leu, M.C., 2013. Additive manufacturing: technology, applications and research needs. Front. Mech. Eng. 8, 215–243.

Hur, S.-M., Choi, K.-H., Lee, S.-H., Chang, P.-K., 2001. Determination of fabricating orientation and packing in sls process. J. Mater. Process. Technol. 112, 236–243.

Kim, D.W., Kim, K.H., Jang, W., Frank Chen, F., 2002. Unrelated parallel machine scheduling with setup times using simulated annealing. Robot. Comput.-Integr. Manuf. 18, 223–231.

Kucukkoc, I., 2019. MILP models to minimise makespan in additive manufacturing machine scheduling problems. Comput. Oper. Res. 105, 58–67.

Lai, K., Chan, J.W., 1997. Developing a simulated annealing algorithm for the cutting stock problem. Comput. Ind. Eng. 32, 115–127.

Leung, S.C., Zhang, D., Sim, K.M., 2011. A two-stage intelligent search algorithm for the two-dimensional strip packing problem. Eur. J. Oper. Res. 215, 57–69.

Li, Q., Kucukkoc, I., Zhang, D.Z., 2017. Production planning in additive manufacturing and 3d printing. Comput. Oper. Res. 83, 157–172.

Li, X., Huang, Y., Tan, Q., Chen, H., 2013. Scheduling unrelated parallel batch processing machines with non-identical job sizes. Comput. Oper. Res. 40, 2983–2990.

Lin, S.-W., Ying, K.-C., 2015. A multi-point simulated annealing heuristic for solving multiple objective unrelated parallel machine scheduling problems. Int. J. Prod. Res. 53, 1065–1076.

Padhye, N., Deb, K., 2011. Multi-objective optimisation and multi-criteria decision making in sls using evolutionary approaches. Rapid Prototyping J. 17, 458–478.

Pandey, P.M., Thrimurthulu, K., Reddy, N.V., 2004. Optimal part deposition orientation in fdm by using a multicriteria genetic algorithm. Int. J. Prod. Res. 42, 4069–4089..

Pour, M.A., Zanardini, M., Bacchetti, A., Zanoni, S., 2016. Additive manufacturing impacts on productions and logistics systems. IFAC-PapersOnLine 49, 1679–1684.

Shahvari, O., Logendran, R., 2017. A bi-objective batch processing problem with dual-resources on unrelated-parallel machines. Appl. Soft Comput. 61, 174–192.

Thrimurthulu, K., Pandey, P.M., Reddy, N.V., 2004. Optimum part deposition orientation in fused deposition modeling. Int. J. Mach. Tools Manuf. 44, 585–594.

Vicari, A., 2015. Advanced applications of 3d printing: from prototypes and parts. Additive Manufacturing for Defence and Aerospace Summit, London. https://additivemanufacturing.iqpc.co.uk/downloads/advanced-applications-of-3d-printing-fromprototypes-and-parts..

Wei, L., Hu, Q., Leung, S.C., Zhang, N., 2017. An improved skyline based heuristic for the 2D strip packing problem and its efficient implementation. Comput. Oper. Res. 80, 113–127.

Wei, L., Zhang, Z., Lim, A., 2014. An adaptive variable neighborhood search for a heterogeneous fleet vehicle routing problem with three-dimensional loading constraints. IEEE Comput. Intell. Mag. 9, 18–30.

Wei, L., Zhang, Z., Zhang, D., Leung, S.C., 2018. A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints. Eur. J. Oper. Res. 265, 843–859.

Wu, S., Kay, M., King, R., Vila-Parrish, A., Warsing, D., 2014. Multi-objective optimization of 3d packing problem in additive manufacturing. IIE Annual Conference. Proceedings, 1485–1494.

Xiao, J., Yang, H., Zhang, C., Zheng, L., Gupta, J.N., 2015. A hybrid lagrangian-simulated annealing-based heuristic for the parallel-machine capacitated lot-sizing and scheduling problem with sequence-dependent setup times. Comput. Oper. Res. 63, 72–82.

Yang, Y., Fuh, J.Y., Loh, H.T., Wong, Y.S., 2003. Multi-orientational deposition to minimize support in the layered manufacturing process. J. Manuf. Syst. 22, 116–129.

Zhang, D., Dong, R., Si, Y.W., Ye, F., Cai, Q., 2018. A hybrid swarm algorithm based on ABC and AIS for 2L-HFCVRP. Appl. Soft Comput. 64, 468–479.

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

Zhang, Y., Bernard, A., Harik, R., Karunakaran, K., 2017. Build orientation optimization for multi-part production in additive manufacturing. J. Intell. Manuf. 28, 1393–1407.