Contents lists available at ScienceDirect

Omega

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

Product sequencing in multiple-piece-flow assembly lines☆

Alena Otto*, Xiyu Li

Department of Management Information Science, University of Siegen, Siegen D-57068, Germany

ARTICLE INFO

Article history:
Received 3 September 2018
Accepted 24 March 2019

Keywords:
Sequencing
Assembly line
Multiple-piece flow
Vector bin packing

ABSTRACT

Modern markets demand mass customization, that is, the manufacture of customized products at low cost. Mass customization represents a major challenge for the organization of assembly lines, which were originally designed for the manufacture of homogeneous products. The multiple-piece-flow assembly line is an organizational innovation that can address this challenge. Here, several customized workpieces, each associated with a separate customer order and, hence, a separate due date, are handled simultaneously in one cycle. Consequently, the idle times decrease as do the manufacturing costs. Multiple-piece-flow assembly lines are used, for instance, in manufacturing industrial equipment.

To the best of our knowledge, this paper is the first to investigate product sequencing in multiple-piece-flow assembly lines. We formalize the underlying planning problem, establish a mixed-integer model, examine its relation to several classic optimization problems, and describe useful problem properties. We leverage these properties to design an effective iterative variable neighborhood heuristic (IVNH). A detailed simulation based on real-world data and the rolling-horizon planning framework confirms that the IVNH is well suited for practical use. Furthermore, extensive computational experiments on well-structured randomly generated data sets show that the IVNH identifies optimal or near-optimal solutions within short run times. It outperformed an off-the-shelf optimization software, and in certain practice settings, the IVNH was even able to substantially reduce average order delays.

1. Introduction

In the coming decades, manufacturing organization is likely to completely change owing to technological innovations and increasingly sophisticated demand. Companies struggle to offer *mass customization* to the clients, that is, the manufacture of customized products at low cost [4,27,64]. Since the assembly line was originally designed for manufacturing homogeneous products, mass customization is particularly challenging for this organizational form. Companies are seeking to re-organize the manufacturing process to secure the advantages of traditional assembly lines, while also being able to produce customized products at low cost.

In traditional paced assembly lines, conveying technology moves products (workpieces) through sequentially arranged stations. At each station, workers or robots process the workpiece for a specified amount of time (i.e., *cycle time*), after which the workpiece is transported to the next station. Paced assembly lines offer several advantages (for more discussion see [16,59]). For instance, manufacturing times can be drastically reduced because of specialization and learning effects [45,61]. Furthermore, paced production increases the transparency of production processes for management and control. If products are not homogeneous, however, large idle times may emerge. Indeed, the processing time of a bottleneck workpiece at a bottleneck station dictates the cycle time. In addition, the processing times of customized products may be very different [15,68]. Therefore, in the manufacture of customized products, the upsurge in cost due to large idle times may offset the cost reduction expected from pacing.

Multiple-piece-flow assembly lines may secure the benefits of paced production and, by having several workpieces bundled together, keep idle times low. In *multiple-piece-flow assembly lines*, a set of workpieces moves together, which enables processing several workpieces at one station in one cycle.

Multiple-piece-flow assembly lines are among several organizational forms that adapt paced assembly lines to the requirements of customized manufacturing. Other organizational forms include, for instance, (i) deploying multi-skilled workers, called *floaters*, that can assist at bottleneck stations (e.g., [32,55]), (ii) increasing the cycle time and assigning several workers to (especially bottleneck) stations (e.g., [28,51,71]), as well as (iii) establishing open-end stations, where workers can cross station borders to keep processing a bottleneck workpiece (cf. mixed-model assembly lines

☆ This manuscript was processed by Associate Editor Segev.
* Corresponding author.
E-mail addresses: alena.otto@uni-siegen.de (A. Otto), xiyu.li@uni-siegen.de (X. Li).
URL: http://www.wiwi.uni-siegen.de/mis/ (A. Otto), http://www.wiwi.uni-siegen.de/mis/ (X. Li)

https://doi.org/10.1016/j.omega.2019.03.013
0305-0483/© 2019 Elsevier Ltd. All rights reserved.

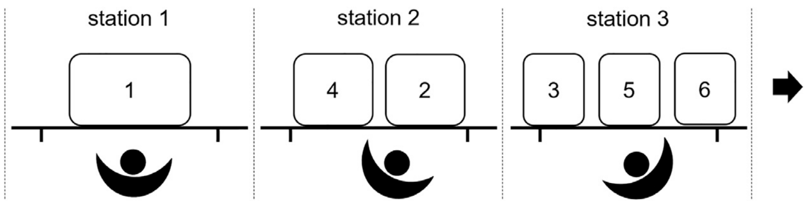


Fig. 1. Illustrative example of a multiple-piece-flow assembly line. Workpieces are transported on mobile platforms between several stations in a series. One or more workpieces may enter the assembly line at the beginning of each cycle. The line is pictured after the launch of the third mobile platform.

with open-end stations in [8,69]). However, it may be difficult to ensure sufficient utilization of the expensive multi-skilled floaters that would keep production costs low. Multi-manned stations may also be impossible because several workers cannot work on certain workpieces without severely hindering each other. In addition, open-end stations may pose hard-to-solve organizational issues of worker coordination and equipment accessibility. Therefore, the multiple-piece-flow assembly line is the first-choice organizational form, or at least the organizational form of interest, for many different companies, based on information that we gathered at cooperation meetings with firms and conferences.

To the best of our knowledge, multiple-piece-flow assembly lines have not yet been discussed in the literature. As a result, the planners and industrial engineers lack evidence-based methodological support in organizing the production process. As a first step in closing this gap, we formulate and study the product sequencing problem in multiple-piece-flow assembly lines in the current paper.

We briefly outline specific examples of multiple-piece-flow assembly lines in Section 1.1 and formulate the respective optimization problem in Section 1.2. Section 1.3 provides an overview of the literature and Section 1.4 explains the article's contribution.

1.1. Examples of multiple-piece-flow assembly lines

In this section, we provide two real-world examples of multiple-piece-flow assembly lines describing two medium-sized companies in separate industries and from different federal states in Germany. Both companies produce customized industrial equipment. Based on interviews with representatives from these and other companies who highlight the advantages of the multiple-piece flow, we suggest that many more companies could benefit from using this alternative for manufacturing customized products.

The first example describes the final assembly of soldering machines shown in Fig. 1. Soldering machines are used in the electronics industry to automatically fix components on circuit boards. Depending on customer requirements, the manufacturing times (and size) of soldering machines may vary significantly. In the final assembly at the company we visited, soldering machines in-process are transported on mobile platforms between different stations. The final assembly is organized as a paced line, where each platform carries one or more workpieces depending on their size and the extent of the manufacturing process required. At each station all the workpieces on one platform have to be processed within the set cycle time. Afterward, the platforms move one station forward.

The second example describes manufacture of industrial jib cranes, which are used in industrial settings, such as warehouses or factory workshops (see Fig. 2). Each workpiece has to be processed at several stations, and each station usually has a machine and a worker to service it. The stations are organized in two lines: a jib line consisting of five stations as well as a pillar line consisting of four stations and a buffer to compensate for different lengths of the two lines. Jibs and pillars are joined together and finally processed in the final station. The production is organized in batches. Each batch (i.e., bundle of workpieces) enters a station

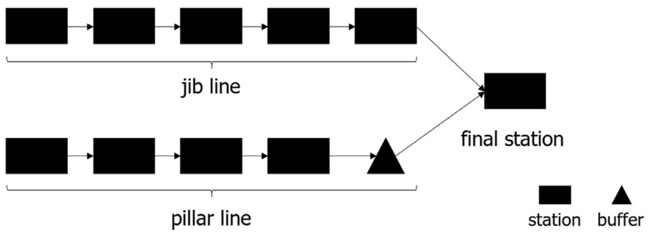


Fig. 2. Schematic illustration of the assembly system for industrial cranes consisting of a pillar line, a jib line, and the final processing station.

at the beginning of the day and all workpieces of that bundle have to be processed by the end of the day. At the beginning of the next day, this bundle of workpieces enters the next station. We discuss this example in more detail in our simulation study in Section 4.3.

1.2. Problem definition

To describe the product sequencing decisions in multiple-piece-flow assembly lines, we formulate the *m-vector bin packing and sequencing problem* (m-BiPacS) as follows. Workpieces $j \in J$ are launched in sets on the paced assembly line to be processed at sequentially arranged stations. Once launched, the composition of a set of workpieces remains the same, i.e., all the workpieces in a set enter and leave each station simultaneously. A set of workpieces is available for a certain amount of time at each station, called *cycle time c*. After c time units are over, the workpieces are moved to the next station and a new set of workpieces is launched at station 1. We denote the set of possible launch times as $\{1, \ldots, \bar{B}\}$, where $\bar{B}$ is the number of potential launch times. Because each workpiece $j \in J$ corresponds to a specific customer order, it has a specific *due date* d_j, which is the latest possible launch time that allows the workpiece to be finished in time, and a specific *release time* r_j, which is the earliest possible launch time when the specification of the product order is fixed and the required material supply is secured. We denote the set of *dimensions* as $\{1, \ldots, m\}$ and the *size* of workpiece $j \in J$ along dimension $i \in \{1, \ldots, m\}$ as v_{ji}. Along each dimension, the total size of the workpieces launched together in a set is restricted. For example, each station represents a separate dimension because at each station the total processing time (= the total size) of workpieces that are launched together cannot exceed the cycle time. In some applications, the size of the mobile platform represents an additional dimension, because the total size of the workpieces that are launched together cannot be larger than the mobile platform. We also introduce cost parameters w_{jb} if workpiece j is launched at time $b \in \{1, \ldots, \bar{B}\}$. For example, if the workpiece is launched after its due date d_j some lateness penalties may be incurred. In this paper, we examine *m-BiPacS* with *non-decreasing costs* in the ordering number of the launch times. That is, for any two launch times $b' > b$, the assignment cost to the second launch time is not lower than the assignment cost to the first launch time: $w_{jb'} \geq w_{jb}\ \forall j \in J$.

The objective of the m-BiPacS is to find mapping $x : J \to \{1, \ldots, \bar{B}\}$ of workpieces to their launch times, i.e., $x(j)$ denotes the

Table 1
Illustrative example: A m-BiPacS instance with six workpieces (jobs), four dimensions $m = 4$, $c = 6$, $r_j = 0\ \forall j \in J$, $\bar{B} = 4$ possible launch times, and total tardiness objective function with $w_{jb} = \max\{b - d_j; 0\}$.

	Workpieces (jobs)						Cost parameters w_{jb}						
								Workpieces (jobs)					
	1	2	3	4	5	6		1	2	3	4	5	6
Processing time at station 1, v_{j1}	5	3	2	3	2	2							
Processing time at station 2, v_{j2}	6	4	1	2	3	2	$b = 1$	0	0	0	0	0	0
Processing time at station 3, v_{j3}	4	2	3	4	2	1	$b = 2$	0	0	0	1	1	1
Size of the workpiece, v_{j4}	3	2	1	2	1	1	$b = 3$	0	1	1	2	2	2
Due date d_j	3	2	2	1	1	1	$b = 4$	1	2	2	3	3	3

launch time of workpiece j. This mapping should be such that

- Each workpiece is launched exactly once and not earlier than its release time: $x(j) \geq r_j\ \forall j \in J$.
- Total size of the workpieces along each dimension does not exceed the cycle time: $\sum_{j \in J: x(j)=b} v_{ji} \leq c \quad \forall b \in \{1, \ldots, \bar{B}\}, \forall i \in \{1, \ldots, m\}$.
- The objective function $F(\mathbf{x}) = \sum_{j \in J} w_{j,x(j)}$ is minimized.

Note that we use bold type, e.g., $\mathbf{x}$, to denote vectors or matrices. For example, in $F(\mathbf{x})$ above, $\mathbf{x} = (x(1), \ldots, x(|J|))$.

Observe that the formulated objective function is quite general and can be used, for example, to minimize the weighted tardiness (by setting $w_{jb} = w_j \cdot \max\{b - d_j; 0\}$ where w_j is the *weight* of workpiece j), weighted lateness (by setting $w_{jb} = w_j \cdot (b - d_j)$), and average completion time (by setting $w_{jb} = b$). Also observe that the m-BiPacS does not prohibit the launch of workpieces after their due date, but it penalizes late completions in the objective function.

Consider an illustrative example with the parameters in Table 1 and a multiple-piece-flow assembly line as shown in Fig. 1. Note that the workpieces that are launched together cannot exceed the size of the mobile platform. We can formulate the respective product sequencing problem as the m-BiPacS. In this formulation, the set $\{1, \ldots, m\}$ describes three assembly stations and an additional dimension, the size of the platform. So that the number of dimensions equals $m = 4$. Parameters v_{j1}, v_{j2}, and v_{j3} denote processing times of workpiece j and parameters v_{j4} denote the length of soldering machine j. Observe that we have normalized v_{j4} in order to set the size of the platform to be equal to 6 as the cycle time. Fig. 1 illustrates a feasible solution for the instance in Table 1 pictured after the launch of the third mobile platform, where workpieces 3, 5, and 6 are launched at time $b = 1$, workpieces 2 and 4 at launch time $b = 2$, and workpiece 1 at launch time $b = 3$. The set of workpieces that are launched together is the *product bundle*, or *bin*. The total processing time of each product bundle in Fig. 1 does not exceed the cycle time at any station; moreover, the total size of each product bundle is not larger than the size of the mobile platform. Because workpiece 4 is launched at time $b = 2$, i.e., 1 period later than its due date, the total tardiness equals 1.

As with the bin packing problem, we can interpret a product bundle launched at time b as an m-dimensional bin with capacity c. Therefore, we refer to product bundles as *bins* in the following sections. We will also use more general term *job* instead of workpiece.

1.3. Literature overview

Assembly lines, in which each workpiece may represent a unique *model* with unique processing times, are referred to as *mixed-model* assembly lines in the literature (see literature reviews on assembly lines and assembly line balancing in [3,6,7,9–11,31,39,60,70]). To the best of our knowledge, none of the article in this thread of the literature considers multiple-piece-flow assembly lines (e.g., [5,26,29,32,56,62,63]). Boysen et al. [12] introduce a classification system for sequencing models in mixed-model assembly lines and provide an extensive literature review. However, this classification solely pertains to one-piece-flow assembly lines.

The m-BiPacS is also closely related to the classical optimization problems of the vector bin packing, multiple knapsack, batch scheduling, and lot-sizing and scheduling. However, the existing problem formulations and their solution methods cannot be straightforwardly applied to solve the m-BiPacS as we explain below.

The *vector bin packing problem (VBP)* revolves around minimizing the number of bins – each stretching in m independent dimensions and having capacity c in each dimension – needed to store all the items $j \in J$ of size v_{ji} along dimension $i \in \{1, \ldots, m\}$. The VBP, which was introduced by [37], is known to be NP-hard in the strong sense. Therefore, most publications study the two-dimensional VBP [2,21,22,42], with the exception of few recent papers mostly motivated by IT applications in which the authors study the VBP with more than two dimensions [20,36,58]. We refer the interested reader to the survey of [24]. The objective function of the m-BiPacS is different from that of the VBP. For instance, as we explain in Section 2.2, solutions of m-BiPacS instances with large numbers of bins may have better objective values that those with the minimal number of bins.

Several studies investigate a generalization of the VBP motivated by logistics applications in which bins represent boxes or containers and the objective is to minimize the sum of bin costs. The cost of the bin may depend on its size [76], the number of items it contains [30], its utilization [52,53], or additional factors such as weight, volume, manpower, and transport distance of the bin [44]. However, these articles usually consider just one- [30,52,53] or two-dimensional bins [44]. Moreover, in contrast to the m-BiPacS, the sequence of bins is irrelevant to the objective function in this generalization of the VBP.

In the *m-dimensional multiple knapsack problem (MMKP)*, a subset of items has to be packed into several multidimensional knapsacks so that the total value of the selected items is maximized [19,48,72]. We refer the interested reader to the surveys of [35,38,48,50,67], and [73]. In contrast to the m-BiPacS, we do not have to pack all the items into the knapsacks (bins) in the MMKP. Moreover, the item profits (=negative cost parameters) $-w_{jb} = -w_j$ are independent from the knapsack's position b, so that the sequence of the knapsacks (=bins) does not influence the objective value.

Similar to the m-BiPacS, the *batching scheduling problem (BP)* is to partition jobs into ordered sets (batches) with respect to some scheduling criterion (e.g., minimize sum of completion times or weighted tardiness) [14]. Two variants of the batching problem have been studied in the literature: the parallel-batching scheduling problem (p-BP) and the serial-batching scheduling problem (s-BP) [13]. The p-BP is motivated by the scheduling of so-called batching machines that can handle several jobs simultaneously

[17,18,74]. However, in the p-BP, the size of a batch, i.e., its processing time, is variable – it is normally computed as the maximum processing time of the jobs in the batch. In the p-BP, the number of jobs in a batch is usually given. Recall that in m-BiPacS, the size of batches, or bins, is given and equals c, whereas the number of jobs in a bin is variable. In the s-BP, the processing time of the batch is equal to the total processing time of all jobs in this batch and some setup time may be required between subsequent batches. The size of the batches (e.g., their processing time) is usually not limited in the s-BP, whereas the restriction on the size of the batches (=total size of the bins along dimensions $\{1,\dots m\}$) is central to the formulation of the m-BiPacS. To the best of our knowledge, only [1,23], and [78] study the s-BP with batches of a limited size, however, even in these cases, the size of a batch is bounded just by the number of jobs. Moreover, none of these papers examines multidimensional batches. We refer the interested reader to the surveys of [13,65,66,75], which explore scheduling problems with batching, and to the survey of [57], which examines various scheduling problems, including the batching problem, in the context of semiconductor manufacturing.

The m-BiPacS is distinct from the *lot-sizing and scheduling problem (LSP)* [33,34,41]. The LSP involves partitioning jobs into batches, or lots, and scheduling production of the batches to meet demand in each production period, while considering trade-offs between the inventory holding costs and the setup costs between the batches. We refer the reader to a recent survey by [25] for a more detailed discussion of the LSP.

1.4. Contribution

To the best of our knowledge, this article is the first to study operations planning in multiple-piece-flow assembly lines. We examine product sequencing decisions. For this purpose, we formulate the m-BiPacS with a specific type of objective functions that include, for instance, weighted tardiness and we propose a mixed-integer model for the m-BiPacS. We show that m-BiPacS is NP-hard in the strong sense. We discuss a polynomially solvable special case of the problem and prove some useful properties, such as the maximum load rule. We also discuss lower bounds and possible ways to strengthen the problem formulation. Our fast and effective iterative variable neighborhood heuristic procedure IVNH finds good-quality solutions for instances of practice-relevant size. In particular, IVNH relies on neighborhood definitions of relatively low computational complexity that are able to construct many distinct neighbor solutions with maximally packed bins (cf. Section 2.2). We perform extensive computational experiments on a large number of randomly generated instances with different settings and perform a simulation based on a real-world data set. For instance, our computational experiments in Sections 4.2 and 4.3 illustrate that idle times can be kept low in multiple-piece-flow assembly lines even with highly customized products. We proceed as follows. In Section 2, we analyze the problem. Section 3 describes our heuristic solution procedure. Computational experiments are presented in Section 4. We conclude with a summary of major findings and future research directions in Section 5.

2. The *m*-vector bin packing and sequencing problem

We present an integer programming formulation for the m-BiPacS in Section 2.1. Afterward, in Section 2.2, we describe several useful problem properties.

2.1. Integer programming formulation

Recall that, in analogy to the bin packing problem, we refer to the possible launch time b of the products in the paced multiple-piece-flow assembly line as *bin b*. The following is a mixed-integer formulation of the m-BiPacS with assignment decision variables:

$$\text{minimize} \quad F(\mathbf{x}) = \sum_{j\in J}\sum_{b=r_j}^{\bar{B}} w_{jb}\cdot x_{jb} \tag{1}$$

$$\text{s.t.} \quad \sum_{b=r_j}^{\bar{B}} x_{jb} = 1 \quad \forall j\in J \tag{2}$$

$$\sum_{j\in J: r_j\le b} v_{ji}\cdot x_{jb} \le c \quad \forall i\in\{1,\dots,m\}, \forall b\in\{1,\dots,\bar{B}\} \tag{3}$$

$$x_{jb}\in\{0,1\} \quad \forall j\in J, \forall b\in\{1,\dots,\bar{B}\}, \tag{4}$$

where binary decision variable x_{jb} denotes whether job j is assigned to bin b ($x_{jb}=1$) or not ($x_{jb}=0$). The objective (1) is to minimize total cost. Parameter $\bar{B}$ describes the upper bound on the number of the required bins, for example, $\bar{B}=|J|+\max_j\{r_j\}$. Each job has to be assigned to exactly one bin and not earlier than its release time r_j (constraints (2)). Capacity constraints (3) should be respected for each bin along each dimension, and constraints (4) state that the decision variables are binary.

2.2. Properties of the m-BiPacS

In this section, we formulate several properties of the m-BiPacS that we use in our solution procedures.

For instance, observe that in contrast to the closely related bin packing problem, optimal solutions of the m-BiPacS instances do not necessarily contain the lowest possible number of bins, although cost parameters w_{jb} in the m-BiPacS are non-decreasing in the ordering number of the bins. Indeed, we may have to increase the number of bins by moving large less-urgent jobs to the last bins and move small urgent jobs to the earlier bins to receive an optimal solution – as illustrated in the example in Table 2. In this example, we can pack all the jobs into two bins as {1, 2}, {3, 4, 5} with cost of 1. But when we move less-urgent job 2 to a later bin and process urgent job 3 earlier as {1, 3}, {2, 4}, {5}, the cost falls to 0.

Lemma 1. *The m-BiPacS is NP-hard in the strong sense.*

Proof. Recall that the VBP is NP-hard in the strong sense [37]. The respective decision problem (D-VBP) is to determine whether a feasible solution with not more than $\bar{B}^{VBP}$ bins exists (yes/no); $\bar{B}^{VBP}$ is a natural number. The decision problem corresponding to the m-BiPacS *(D-m-BiPacS)* is to find out whether a feasible solution with an objective value of not more than $\bar{F}$ exists (yes/no); $\bar{F}$ is some real number. Observe that the D-VBP can be solved by solving the instance of the D-m-BiPacS with parameters $w_{jb}=0$ if $b\le d_j$ and $w_{jb}=1$ otherwise, $r_j=0, d_j=\bar{B}^{VBP}$, and $\bar{F}=0$. Since D-VBP is well-known to be strongly NP-complete [37], we conclude the D-m-BiPacS is strongly NP-complete as well and m-BiPacS is strongly NP-hard. □

Table 2
Illustrative example: A m-BiPacS instance with five jobs, one dimension $m=1$, $c=10$, $r_j=0\ \forall j\in J$, and total tardiness objective function $w_{jb}=\max\{b-d_j;0\}$.

	Jobs				
	1	2	3	4	5
Size v_{j1}	5	5	3	3	4
Due date d_j	1	2	1	2	3

Let $J^b(s)$ be the set of jobs assigned to bin b and $J^{<b}(s)$ be the set of jobs assigned to the bins preceding bin b in some solution s of an m-BiPacS instance. In the following, we simply write J^b and $J^{<b}$ if the notation can be unambiguously constructed from the context. We call bin b *maximally packed* if no additional job $j \in J \backslash (J^{<b} \cup J^b)$ can be assigned to this bin without violating either its release time r_j or the bin's capacity.

Property 1. *(Maximum load rule) There is an optimal solution to the m-BiPacS with only maximally packed bins.*

Proof. The claim follows immediately from the assumption of non-decreasing cost parameters w_{jb} in the ordering number of the bins (cf. the definition of w_{jb} in Section 1.2). □

The maximum load rule is well known in bin packing. It allows us to construct algorithms that examine *only* solutions with maximally packed bins.

We use the next property, Property 2, to construct a well-performing neighborhood (neighborhood $\mathcal{N}_3$) in our heuristic algorithm in Section 3 and to strengthen the model formulation (1)–(4), as described below.

Property 2. *The m-BiPacS with equal job sizes (i.e., $v_{ji} = v_{j'i}$ $\forall j, j' \in J$ and $\forall i \in \{1, \ldots, m\}$) is polynomially solvable.*

Proof. Let n_b be the maximal number of jobs that can be packed in bin b. To assign jobs to bins, we formulate a transportation problem [see [43,77]] as described below. The transportation problem can be solved, for example, with the algorithm of [49] in polynomial time.

The transportation problem consists of finding the number of items to be transported from each supply center to each demand center in order to satisfy the demand, so that the transportation cost is minimized and the needs of demand centers are satisfied. We model $\bar{B}$ bins as demand centers with demand n_b (because each bin contains a maximum of n_b jobs) and jobs as supply centers with a supply of 1 each (because each job has to be assigned to exactly one bin). We introduce $\bar{B} \cdot n_b - |J|$ dummy jobs to match the supply and the demand. We set transportation costs of one product unit from supply center $j \in J$ to demand center b to w_{jb} if $b \geq r_j$ and to some sufficiently large number M, e.g., $M = \sum_{j \in J} \max_{b \in \{1,\ldots,\bar{B}\}} \{w_{jb}\} + 1$, otherwise. Observe that each job j has to be assigned to exactly one bin and, therefore, the objective value of any feasible solution, in which all jobs are assigned to bins not earlier than their release times, is lower than M. Transportation costs per product unit from dummy supply centers equal 0.

The constructed transportation problem indeed solves the m-BiPacS with equal job sizes, because it respects all the constraints of the m-BiPacS (and only them): bin capacity constraints (3) as well as assignment and release time constraints (2) (see Section 2.1). The constructed problem also correctly represents the objective function of the m-BiPacS.

Note that if jobs additionally have the same assignment costs ($w_{jb} = w_b$ $\forall j \in J$), we can get an optimal solution by examining them in the non-decreasing order of their due dates and assigning them to the earliest bin possible. □

Observe that the bin packing problem is closely related to the m-BiPacS, for instance, the decision problem of the bin packing is a special case of the decision problem D-m-BiPacS. Therefore, several properties of the bin packing problem can be directly adapted to the m-BiPacS. These properties include, for example, the Jackson dominance rule [cf. [46]] and the lower bounds of [54]. However, in our experience, the quality of these lower bounds drastically deteriorates with the increasing number of bin dimensions m. Therefore, we formulate an alternative lower bound LB based on computing N_b, which denotes an upper bound on the number of jobs that can be packed into bin b in a solution with maximally packed bins due to bin size and release time restrictions, and relaxing the problem to the transportation problem. We describe lower bound LB below. Further, observe that model (1)–(4) can be strengthened by adding the following constraints as a simple and computationally inexpensive way to strengthen model formulation (1)–(4):

$$\sum_{j \in J: r_j \leq b} x_{jb} \leq N_b \qquad \forall b \in \{1, \ldots, \bar{B}\} \tag{5}$$

In the computational experiments in Section 4.2.3, we show that constraints (5) may speed up the standard solver by several times.

We compute N_b in the increasing order of the bin numbers starting with $N_{b'}, b' = \min_{j \in J}\{r_j\}$. We estimate N_b by solving the *one*-dimensional unit-profit knapsack problem for the current set of jobs J' with *surrogate constraints* (cf. [21,40]), that is, some weighted sums of constraints (3). The objective $KS(\mathbf{x})$ of the knapsack problem is to maximize the number of the selected jobs. Given some weights $l_i \geq 0, \sum_{i=1}^{m} l_i = 1$, the knapsack problem for bin b can be notated as follows:

$$\text{maximize} \qquad KS(\mathbf{x}) = \sum_{j \in J'} x_{jb} \tag{6}$$

$$\sum_{i=1}^{m} \sum_{j \in J'} l_i \cdot v_{ji} \cdot x_{jb} \leq c \cdot \sum_{i=1}^{m} l_i = c \tag{7}$$

$$x_{jb} \in \{0, 1\} \qquad \forall j \in J' \tag{8}$$

We set $N_b = KS(\mathbf{x})$. Observe thatsack problem simply by assigning the jobs in the non-decreasing order of their surrogate size in $O(|J| \log |J|)$.

Initially, for $N_{b'}$, we assign all jobs with sufficiently early release times to set $J' = \{j \in J : r_j \leq b'\}$. Because of the maximal load-rule, bin b' will contain *at least one job* if set J' is not empty. Therefore, for computing $N_{b'+1}$, we ignore job $j' \in J'$ with the largest surrogate size and update set $J' = \left(J' \setminus \{j'\}\right) \cup \{j \in J : r_j = b' + 1\}$ as described in Algorithm 1. Indeed, since we can solve the knapsack

Algorithm 1: Computation of N_b.

```
1  $b' := \min_{j \in J}\{r_j\}$;
2  $J' := \{j \in J : r_j \leq b'\}$;
3  for $\left(b := b'; b \leq \bar{B}; b{+}{+}\right)$ do
4      if $|J'| > 0$ then
5          Find $N_b$ by solving knapsack problem~(6)–(8) for $J'$ and $b$;
6          Set $j'$ to be a job in $J'$ with the largest (surrogate) size;$J' := \left(J' \setminus \{j'\}\right) \cup \{j \in J : r_j = b + 1\}$;
7      else
8          $N_b := 0$;
9          $J' := J' \cup \{j \in J : r_j = b + 1\}$;
10 .
```

problem (6)-(8) by assigning the jobs in the non-decreasing order of their surrogate size, $N_{b'+1}$ is the largest if we select job $j' \in J'$ with the largest surrogate size. We proceed in a similar manner for all $b \in \{b', \ldots, \bar{B}\}$.

Based on the calculated parameters N_b, we compute lower bound LB by formulating and solving a transportation problem with $n_b = N_b$, as described in the proof of Property 2.

Observe that LB is indeed a lower bound for the m-BiPacS because each feasible solution of the m-BiPacS with maximally packed bins (see Property 1) is also feasible for the formulated transportation problem and has the same objective value. As explained above, N_b is an upper bound on the number of jobs that can be assigned to bin b in a feasible solution with maximally packed bins. Therefore, in fact, the formulated transportation problem is equivalent to problem (1)–(4) with constraints (3) relaxed to $\sum_{j\in J: r_j \le b} x_{jb} \le N_b \; \forall b \in \{1, \ldots, \bar{B}\}$.

3. Iterative variable neighborhood heuristic

To address large problem instances of the m-BiPacS, we propose an iterative variable neighborhood heuristic *(IVNH)*. The IVNH examines only solutions with *maximally packed bins*. Recall that the objective value of any solution does not get worse and potentially significantly improves, if we shift jobs to pack bins maximally (cf. Section 2, Property 1). This is a key factor influencing the performance of the IVNH.

The IVNH iteratively performs local search with three different neighborhoods: $\mathcal{N}_1, \mathcal{N}_2$, and $\mathcal{N}_3$ (see Section 3.2). Having reached a local optimum with respect to neighborhood $\mathcal{N}_i$, $i = 1, 2$, we move to the next neighborhood $\mathcal{N}_{i+1}$. Whenever we reach a local optimum in all the three neighborhoods, we perform a diversification procedure (see Section 3.3). We construct our initial solution with a greedy procedure as described in Section 3.1. Section 3.4 provides a summary of the IVNH and its pseudocode.

3.1. Initial heuristic

We adapt the priority rule based method *(PRBM)* of [61] to find an initial solution. The PRBM is a greedy procedure based on multiple cues, which constructs a large number of different good-quality feasible solutions. At each PRBM iteration (l), the PRBM constructs a feasible solution by setting priorities $p_j^{(l)}$ for each job $j \in J$ and assigning jobs to bins in a non-increasing order of their priorities. We apply a *first-fit* procedure to assign jobs, i.e., we assign each job to the earliest bin, to which it can be assigned because of bin size and release time restrictions.

We compute three *elementary* priority values for each job j scaled to the interval of [0,1]: p_j^w, p_j^{dd}, and p_j^s. Priority values describe factors (cues) that are likely to lead to a good assignment of jobs. Values $p_j^w = \frac{w_j}{\max_{j' \in J}\{w_{j'}\}}$ force jobs with large weights to be assigned to early bins. Values p_j^{dd} assign high priority values to jobs with early due dates. We set $p_j^{dd} = 1 - \frac{d_j - \min_{j' \in J}\{d_{j'}\}}{\max_{j' \in J}\{d_{j'}\} - \min_{j' \in J}\{d_{j'}\}}$ if due dates differ among jobs (i.e., $\max_{j' \in J}\{d_{j'}\} > \min_{j' \in J}\{d_{j'}\}$) and set $p_j^{dd} = 0$ otherwise. Value p_j^s is a measure of the job's size: $p_j^s = \frac{\sum_{i=1}^m v_{ji} \cdot e_i}{\max_{j' \in J}\{\sum_{i=1}^m v_{j'i} \cdot e_i\}}$, where coefficient $e_i := \sum_{j'' \in J} v_{j''i}$ assigns a higher importance to the bottleneck dimensions.

We compute priorities as a weighted sum of the three elementary priority values: $p_j^{(l)} = \pi_1^{(l)} \cdot p_j^w + \pi_2^{(l)} \cdot p_j^{dd} + \pi_3^{(l)} \cdot p_j^s$. Coefficients $\pi_1^{(l)}, \pi_2^{(l)}, \pi_3^{(l)}$ weigh the importance of elementary priority values. We exploit the low computational complexity of priority rules and perform a brute search over many possible values of the elementary priority values' weights. We select values $\pi_1^{(l)}$ and $\pi_2^{(l)}$ from the set $\{0, 0.1, 0.2, \ldots, 1\}$ to assign high importance to jobs with large weights and early deadlines. Observe that ranking jobs according to their size has two opposing effects. From the one hand, if we assign larger jobs first, we are likely to require less bins, which is good. On the other hand, since assignment costs are monotonous non-decreasing in the ordering number of the bins, we are interested in assigning as many jobs as possible to the earlier bins to reduce the assignment costs for the largest possible number of jobs. For the latter reason, it may be beneficial to assign smaller jobs first. Therefore, we construct priorities both with

Table 3
Instance of the m-BiPacS with eight jobs, three dimensions, $c = 10$, equal release times $r_j = 0 \; \forall j \in J$, and the objective to minimize total tardiness.

	Jobs							
	1	2	3	4	5	6	7	8
v_{j1}	5	4	3	4	2	7	2	3
v_{j2}	4	5	4	2	1	6	3	2
v_{j3}	2	4	5	5	3	7	1	1
d_j	1	1	1	2	2	3	3	4

Table 4
Priorities of the jobs in the instance of Table 3 according to the EDD rule ($\pi_1^{(l)} = 0, \pi_2^{(l)} = 1$, and $\pi_3^{(l)} = 0$).

	Jobs							
	1	2	3	4	5	6	7	8
Priority values $p_j^{(l)}$	1	1	1	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	0

non-negative and negative values of coefficient $\pi_3^{(l)}$ and select values $\pi_3^{(l)}$ from the set $\{-1, -0.9, -0.8, \ldots, 1\}$.

We perform $11 \cdot 11 \cdot 21 = 2541$ iterations of the PRBM in total, because $\pi_1^{(l)}$ and $\pi_2^{(l)}$ take 11 values and $\pi_3^{(l)}$ takes 21 values. Since some of these iterations use equivalent priorities, such as priorities with $(\pi_1^{(l)}, \pi_2^{(l)}, \pi_3^{(l)})$ of (0.1,0.1,0.1) and (0.2,0.2,0.2), the number of the constructed distinct feasible solutions is somewhat lower. We choose the best found solution to be the initial solution.

Consider the instance in Table 3 with eight jobs, three dimensions, $c = 10$, equal release times $r_j = 0 \; \forall j \in J$, and the objective to minimize total tardiness. Table 4 illustrates priorities of the jobs in some iteration l with $\pi_1^{(l)} = 0, \pi_2^{(l)} = 1$, and $\pi_3^{(l)} = 0$. (We call this rule as the *earliest due date rule (EDD)* in our computational experiments.) We break the ties by taking a job with a lower ordering number. In this iteration l of the PRBM, we receive a feasible solution as follows. We sort jobs in a non-decreasing order of their priorities as (1,2,3,4,5,6,7,8) and assign them with the first-fit procedure to receive solution {1, 2}, {3, 4}, {5, 6}, {7, 8} with objective value of 3.

The time complexity of the first-fit assignment procedure is $O(|J| \cdot m \cdot \bar{B})$ because we have to examine m dimensions of up to $\bar{B}$ bins for each job. The time complexity to compute vectors of elementary priority values is linear in $|J|$ for p^w and p^{dd}. It is linear in $|J| \cdot m$ for p^s. Therefore, the time complexity of one iteration of the PRBM equals $O(|J| \cdot m \cdot \bar{B})$, or $O(|J|^2 \cdot m)$, because the number of non-empty bins cannot exceed the number of jobs.

3.2. Local search procedures

We formulate three neighborhoods: $\mathcal{N}_1$ is based on two-job exchanges, $\mathcal{N}_2$ examines job reinsertions, and $\mathcal{N}_3$ looks for alternative sequences of the bins. We formulate neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ such that they *(i)* examine only solutions with maximally packed bins, *(ii)* have moderate time complexity, and *(iii)* each feasible solution has, as a rule, many distinct neighbors. Although the resequencing neighborhood $\mathcal{N}_3$ may contain solutions with not maximally packed bins, the maximum load rule is enforced in the further search since we always perform $\mathcal{N}_1$- and $\mathcal{N}_2$-local search for such solutions.

We construct a neighbor solution s' of the incumbent solution s *indirectly* by recoding solution s as shown in Fig. 3(i). We represent incumbent solution s as a *permutation of jobs* $\sigma(s)$ skipping the assignment of jobs to the bins (jobs within each bin can be arbitrarily ordered). Afterward, we perform a neighborhood move and receive another permutation of jobs $\sigma(s')$. Then we construct

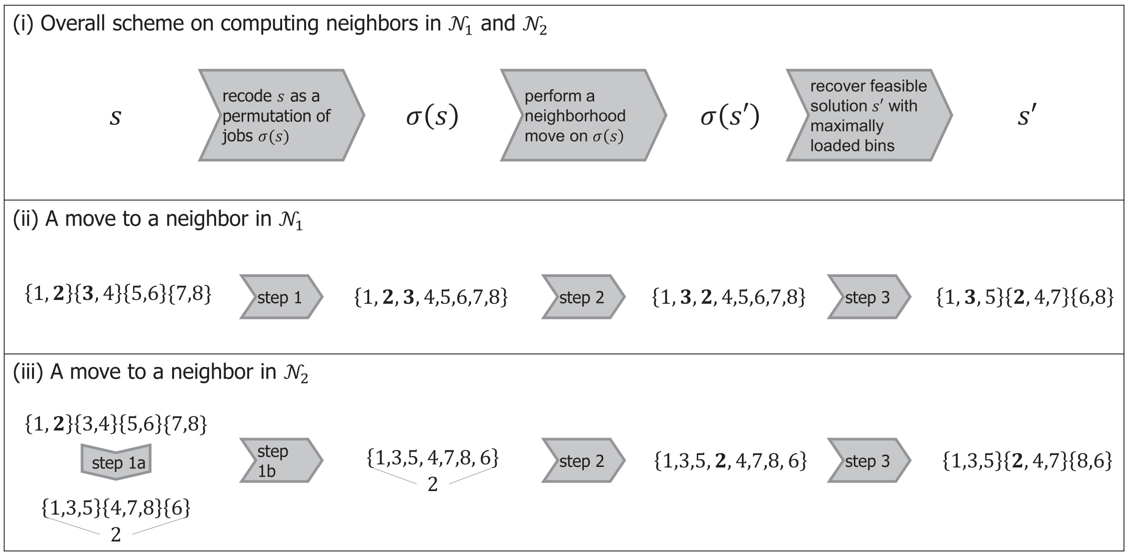

(i) Overall scheme on computing neighbors in $\mathcal{N}_1$ and $\mathcal{N}_2$
s
recode s as a permutation of jobs $\sigma(s)$
$\sigma(s)$
perform a neighborhood move on $\sigma(s)$
$\sigma(s')$
recover feasible solution s' with maximally loaded bins
s'
(ii) A move to a neighbor in $\mathcal{N}_1$
{1, **2**}{**3**, 4}{5,6}{7,8}
step 1
{1, **2**, **3**, 4,5,6,7,8}
step 2
{1, **3**, **2**, 4,5,6,7,8}
step 3
{1, **3**, 5}{**2**, 4,7}{6,8}
(iii) A move to a neighbor in $\mathcal{N}_2$
{1, **2**}{3,4}{5,6}{7,8}
step 1a
{1,3,5}{4,7,8}{6}
2
step 1b
{1,3,5, 4,7,8, 6}
2
step 2
{1,3,5, **2**, 4,7,8, 6}
step 3
{1,3,5}{**2**, 4,7}{8,6}

Fig. 3. Illustration: The computation of neighbors in $\mathcal{N}_1$ and $\mathcal{N}_2$.

s' by assigning jobs of $\sigma(s')$ to bins with the first-fit procedure as explained in Section 3.1. Such indirect local search procedures have two major advantages compared to the respective conventional local search procedures. First of all, if we would perform neighborhood moves on the original (not recoded) solution s, the number of these moves would be significantly limited, since we will have to preserve the feasibility of the resulting neighbor solution. In our indirect local search procedures, the neighbor moves are performed on permutation of jobs $\sigma(s)$ so that we are not restricted by the m-BiPacS constraints at this step and any resulting permutation of jobs is 'feasible'. We construct a feasible solution afterward with very flexible first-fit procedure. Secondly, our indirect local search procedures always result in solutions with maximally packed bins. We provide examples for these statements and explain Fig. 3(ii) and (iii) below.

Two-job exchange neighborhood $\mathcal{N}_1$. Let denote the incumbent solution as s. Let $j \in J$ be one of jobs and $J^b \subseteq J$ be the set of jobs assigned to the same bin as j in s. We construct a permutation of jobs $\sigma(s)$ by skipping the assignment of jobs to the bins (step 1). Given $\sigma(s)$, we receive a neighbor s' of s by swapping j with some job $j' \in J \backslash J^b$ in $\sigma(s)$ (step 2) and assigning jobs to bins in the resulting order by the first-fit procedure (step 3).

Consider the instance in Table 3 and $s = \{1, 2\}\{3, 4\}\{5, 6\}\{7, 8\}$ as initial solution. Fig. 3(ii) illustrates how to construct a neighbor of solution s by swapping jobs 2 and 3. We construct a permutation of jobs $\sigma(s) = \{1, 2, 3, 4, 5, 6, 7, 8\}$ in step 1. After swapping jobs 2 and 3, we get permutation $\sigma(s') = \{1, 3, 2, 4, 5, 6, 7, 8\}$ (step 2). Then, we apply the first-fit procedure, i.e., we assign each job to the earliest bin, to which it can be assigned, considering them in the order provided by $\sigma(s')$ (step 3). For instance, jobs 1 and 3 can be assigned to bin 1, afterward, jobs 2 and 4 can only be assigned to bin 2. The next job, job 5, fits bin 1; and so on. The resulting solution s' is, in fact, an optimal solution with objective value of 1.

The time complexity to examine all the neighbors of solution s is $O(|J|^3 \cdot m \cdot \bar{B})$ because there are up to $O(|J|^2)$ job exchanges and the complexity of first-fit procedure is $O(|J| \cdot m \cdot \bar{B})$.

Let us illustrate the effectiveness of the formulated indirect neighborhood moves by comparing neighborhood $\mathcal{N}_1$ to the conventional two-job exchange neighborhood on the example in Table 3 for $s = \{1, 2\}\{3, 4\}\{5, 6\}\{7, 8\}$. In $\mathcal{N}_1$, solution s has 19 distinct neighbor solutions and all of them consist of maximally packed bins. In the conventional neighborhood search, in which we examine feasible swaps of two jobs, there are only 16 (15% less) distinct neighbor solutions and only four of them consist of maximally packed bins; none of these solutions are optimal. This illustrative example is not an exception (see Appendices A and B). It can be shown that neighborhood $\mathcal{N}_1$ of some feasible solution s contains *all* the neighbors of the conventional two-job exchange neighborhood of s that are improved by the subsequent shift of jobs to receive maximally packed bins (see Appendix A). Recall that by shifting jobs to earlier bins we cannot worsen, but can potentially improve the objective function (see Property 1 in Section 2.2).

Job reinsertion neighborhood $\mathcal{N}_2$. Let consider reinsertion of job $j \in J$. Observe that the recoding of a solution is a one-to-many mapping because several different permutations of jobs may correspond to a single solution of m-BiPacS. Therefore, we introduce an additional computational step of low time complexity to exclude (almost all the) identical solutions from the neighborhoods: We transform incumbent solution s into partial solution s^p by assigning jobs $J \backslash \{j\}$ from $\sigma(s)$ according to the first-fit procedure (step 1a). Then, we perform a neighborhood move by inserting job j between two consequent neighboring jobs j_{k-1} and j_k in $\sigma(s^p)$, which belong to two *different* bins in s^p (step 1b and step 2). Afterward, we assign jobs to bins according to the first-fit procedure to receive neighbor solution s' (step 3).

For the instance in Table 3 and initial solution $s = \{1, 2\}\{3, 4\}\{5, 6\}\{7, 8\}$, we can construct a neighbor solution by reinserting job 2 as shown in Fig. 3(iii). We receive an optimal solution s' with objective value of 1.

The time complexity to examine all the neighbors of some solution s is $O(|J|^3 \cdot m \cdot \bar{B})$ because there are $|J|$ candidates for the job reinsertion that can be reinserted in up to $O(|J|)$ positions and the complexity of the first-fit procedure is $O(|J| \cdot m \cdot \bar{B})$.

Similar to neighborhood $\mathcal{N}_1$, neighborhood $\mathcal{N}_2$ constructs many distinct neighbor solutions with maximally packed bins. Recall that in the conventional job reinsertion neighborhood, we construct *feasible* neighbor solutions by shifting some job j to some other bin. Therefore, in this conventional neighborhood of some solution s with *maximally packed bins*, we can move any job j only to a later bin. Thus, there exist no improving neighbors for solutions with maximally packed bins.

Overall, in our local search procedure, we search neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ of incumbent solution s until we find some neighbor s'

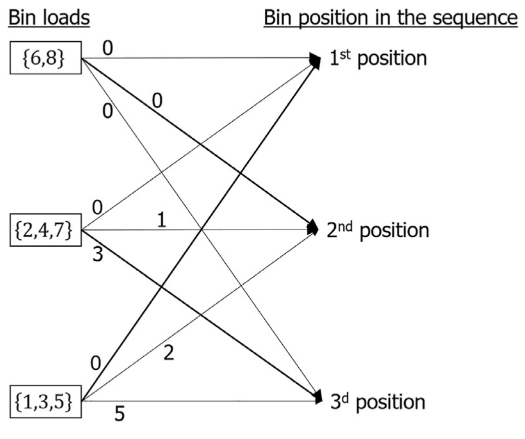


Fig. 4. Illustration: The computation of neighbors in $\mathcal{N}_3$.

with a better objective value (a so-called *improving neighbor*). We immediately set the improving neighbor to be a new incumbent solution $s = s'$. We repeat our search, until the incumbent solution does not have any improving neighbors, i.e., until a local optimum is found.

Resequencing neighborhood $\mathcal{N}_3$. In the third neighborhood, $\mathcal{N}_3$, we treat the partition of jobs $J = J^1 \cup J^2 \cup \ldots$ into bin loads as given and look for a permutation of the bin loads that results in a feasible solution with the best possible objective value. We formulate the respective assignment problem, in which we assign the bin loads to specific launch times, and solve it in $O(\bar{B}^3 + |J| \cdot \bar{B})$ because the improved version of the Hungarian algorithm takes $O(\bar{B}^3)$ [47] and we can construct the assignment graph in $|J| \cdot \bar{B}$.

Consider neighborhood $\mathcal{N}_3$ of $s = \{1,3,5\}\{2,4,7\}\{6,8\}$. Fig. 4 visualizes the resulting assignment problem. Its optimal solution is 1. Therefore, there is no better assignment (sequence of bins) and s is a local optimum with respect to $\mathcal{N}_3$.

3.3. Diversification procedure

A good diversification procedure would overcome a local optimum by jumping to a *far-enough* solution in the solution space. And it should also learn some “good” characteristics of the incumbent solution in order to jump to a “good” region in the solution space. In our computational experiments, the diversification procedure we describe below has outperformed the conventional random diversification, which is jumping to a randomly generated solution (cf. Section 4.2.2). Presumably, our diversification procedure is more likely to keep some good combinations of jobs from the current local optimal solution s^* together in the same bin.

To perform diversification, we randomly assign jobs of the recoded current local optimal solution $\sigma(s^*)$ to f categories (subsequences of jobs), i.e., we randomly generate a category for each job from discrete uniform distribution $\mathcal{U}\{1, f\}$ (step 1 and step 2 in Fig. 5). Afterward, we form a new sequence of jobs $\sigma(s)$ by taking jobs with number one from the f categories, then jobs with number two and so on (step 3). We receive a new incumbent solution s by assigning jobs to bins in the resulting order $\sigma(s)$ by the first-fit procedure (step 4).

Fig. 5 illustrates the diversification procedure for local optimal solution $s^* = \{1,3,5\}\{2,4,7\}\{6,8\}$ and $f = 2$ categories for the instance in Table 3. In step 1, we receive a permutation of jobs $\sigma(s^*) = \{1,3,5,2,4,7,6,8\}$. A random assignment of jobs to the categories resulted in jobs 1 and 6 belonging to category 2 and the rest of the jobs belonging to category 1 (step 2). After having alternately assigned the jobs from the two categories, we receive a new sequence of jobs $\sigma(s) = \{3,1,5,6,2,4,7,8\}$ (step 3). And the first-fit procedure constructs solution $s = \{3,1,5\}\{6,7\}\{2,4\}\{8\}$ (step 4). Observe that the resulting solution s preserves a well-fitting bin load {1, 3, 5}.

3.4. Summary of the IVNH procedure

This section summarizes the IVNH procedure, which uses parameters *Tlim* (time limit), *Glim* (limit on the number of non-improving iterations), *Flim* (parameter of the diversification procedure), and f_{init} (parameter of the diversification procedure). We describe the parameter values that we used in our experiments in Section 4.1.

Let denote the following steps as an *iteration* of the IVNH: construction of a new incumbent solution (either as described in Section 3.1 or as described in Section 3.3) and local search until a local optimal solution with respect to the three neighborhoods $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$ is found.

We start by initializing iterations counter *it*, the objective value of the best solution found so far F^*, the number of iterations from the last update of the best found objective value *gt*, the locally optimal solution found in the previous iteration s^*_d, and diversification parameter f (row 1). The IVNH starts with an initial solution computed as described in Section 3.1 (row 2). Afterward, we perform the local search with respect to the three neighborhoods $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$ until a feasible solution s^*_3 is found, which is locally optimal with respect to all the neighborhoods (rows 3–8). In the diversification procedure, we initialize parameter f as $f = f_{init}$ (row 1). If the objective function of the new local optimum s^*_3 in current iteration *it* is not better than that of the previous one s^*_d, we return to the previous local optimal solution s^*_d and increase f by one (row 13) unless $f = Flim$. In the latter case, we accept the current local optimal solution as incumbent solution $s^*_d = s^*_3$ and reset $f = f_{init}$ (row 11). We update $s^*_d = s^*_3$ and reset the counter f otherwise (row 15). At each iteration, we update the objective value of the best solution found so far (rows 16–19). We stop the IVNH either when the time limit *Tlim* have been reached or the maximal number of iterations without the improvement in the best found

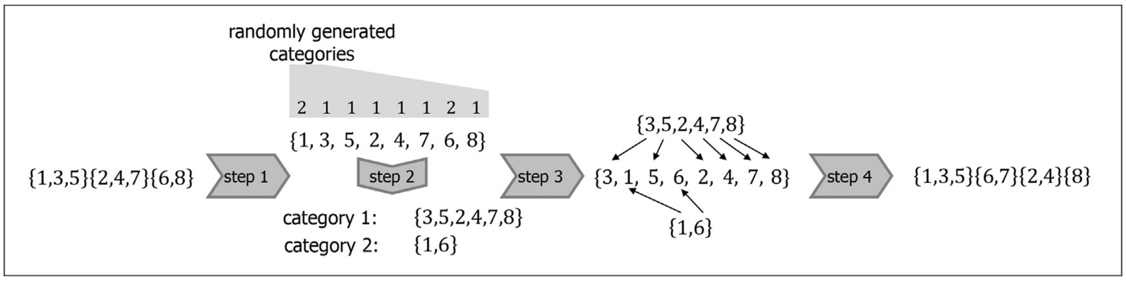


Fig. 5. Illustration of the diversification procedure.

solution equals *Glim* (rows 23–25). For the pseudocode of the IVNH procedure see Algorithm 2.

Algorithm 2: Pseudocode of the IVNH.

```
    // initialization
 1  Initialize $f := f_{init}, F^* := \infty$, $gt := 0$; $s^*_d := \emptyset$; $it := 0$;
 2  Construct $s$ with initial heuristic (see Section 3.1);
    // local search
 3  Starting from $s$, find locally optimal solution $s^*_1$ with respect to $\mathcal{N}_1$;
 4  Starting from $s^*_1$, find locally optimal solution $s^*_2$ with respect to $\mathcal{N}_2$;
 5  Starting from $s^*_2$, find locally optimal solution $s^*_3$ with respect to $\mathcal{N}_3$;
 6  if $s \neq s^*_3$ then
 7      $s := s^*_3$;
 8      go to 3;
    // diversification
 9  if $s^*_d \neq \emptyset$ and $F(s^*_d) \leq F(s^*_3)$ then
10      if $f = Flim$ then
11          $f := f_{init}$; $s^*_d := s^*_3$;
12      else
13          $f := f + 1$; $s^*_d := s^*_d$;
14  else
15      $f := f_{init}$; $s^*_d := s^*_3$;
16  if $F^* > F(s^*_d)$ then
17      $F^* := F(s^*_d)$; $gt := 0$;
18  else
19      $gt := gt + 1$;
20  $it := it + 1$;
21  Apply diversification procedure for $s^*_d$ with parameter $f$ to receive $s^*_{div}$ (see Section 3.3);
22  $s := s^*_{div}$;
    // verification of the stopping criterion
23  if elapsed time $< Tlim$ and $gt < Glim$ then
24      goto 3;
25  .
```

4. Computational experiments

We perform our computational experiments on a diversified randomly generated data set and as a simulation based on the real-world data of an industrial equipment producer. Section 4.1 describes the randomly generated data sets. We then examine the performance of the IVNH and of an off-the-shelf solver in Section 4.2 and illustrate the results of the IVNH in a simulation of a real-world rolling-horizon planning framework in Section 4.3.

4.1. Data generation

Since the literature lacks a benchmark data set for the *m*-BiPacS, we have randomly generated a data set with a wide range of various instance characteristics for our experiments. We examine a total of 440 instances in our experiments, encompassing 22 parameter settings with 20 randomly generated instances for each parameter setting.

The *large data set (LDS)* contains large instances with $n \geq 100$ jobs. In the *basic* setting, we set $n = 150$, number of dimensions $m = 5$, dimensions' capacities $c = 100$, and release times $r_j = 0$ for all the jobs. For each job $j \in J$ and each dimension $i \in \{1, \dots, m\}$, we randomly and independently generate sizes v_{ji} from the interval $[\underline{v}, \overline{v}] = [10, 30]$. Deadlines d_j are randomly and independently drawn from the interval $[\lceil \frac{L}{2} \rceil, \lceil L \rceil]$ (we designate this interval as *large*), where L is a reference value that depends on the number of jobs and their sizes. The value of L increases if the number of jobs or their sizes increase: $L = \frac{n}{c/E(v_{ji})}$, where $E(v_{ji})$ is the expectation of the job's size. In the basic setting, $[\lceil \frac{L}{2} \rceil, \lceil L \rceil] = [15, 30]$. We have selected this interval for the due dates to include both in-time and delayed jobs in each instance. For example, on average about 5.5% of jobs are delayed in the best known solutions of the basic setting. We minimize the total tardiness, so that $w_{jb} = \max\{b - d_j; 0\}$.

We construct another 11 settings in the LDS by varying one set of parameters at a time. For example, we examine two additional settings with $n = 100$ and $n = 200$, two settings with $m = 2$ and $m = 8$, and four settings with intervals $[\underline{v}, \overline{v}]$ equal to [0,20], [20,40], [30,50], and [0,100]. We introduce a setting with release times by randomly and independently drawing them from $[0, \lceil \frac{L}{2} \rceil]$. Two further settings examine smaller, more restrictive due dates by drawing them from intervals $[0, \lceil \frac{L}{2} \rceil]$ *(small)* and $[\lceil \frac{L}{4} \rceil, \lceil \frac{3L}{4} \rceil]$ *(medium)*. The LDS contains 12 settings with 240 instances in total.

The *small data set (SDS)* contains smaller instances with $n = 30$ jobs. Because we fix the number of jobs in the SDS, it consists of 10 settings arranged along the same lines as the LDS.

In the following sections, we describe a setting as ($n, m, [\underline{v}, \overline{v}]$, release time, due date).

We performed our experiments on a personal computer with an Intel i7-8700K processor, 6 cores, and 16 GB RAM. We compared the performance of the IVNH to that of the off-the-shelf software IBM ILOG CPLEX 12.8 (which we simply call *CPLEX* below). We used default settings of CPLEX. We set the objective function to optimize the total tardiness, i.e., $w_{jb} = \max\{b - d_j; 0\}$, unless specified otherwise.

We set the IVNH parameters as follows. We selected a rather large limit for nonimproving iterations, $Glim = 20$, since when the incumbent solution has not been improved in the 20 subsequent iterations, the odds of finding a better solution appear to be very low. The diversification parameter $f_{init} = 2$ allows to try a moderate degree of diversification with $f = 2$ categories first and parameter $Flim = 4$ enables a high degree of diversification, which constructs a new incumbent solution far enough away from the local optimum. We set $Tlim = 10$ minutes, which is an acceptable time limit in practical applications.

In the computational experiments, we report the *average absolute performance* and the *average relative performance*. Because optimal objective values of some of the instances are not known, we compare the algorithm's solutions for each instance ψ to the *best* objective value found in all the computational experiments for this instance ψ.

The average absolute performance refers to the average difference in the absolute objective value of the algorithm's solution and the objective value of the best found solution. For instance, the absolute performance equals 1 if just one tardy job was assigned one bin later compared to the best found solution.

The average relative performance denotes the average relation of the difference between the objective values found by the algorithm and the best found solution to the best found objective value. Note, however, that the average relative performance is generally *not very informative* for such objective functions as tardiness. Overall, relative performance may be large even for very small absolute gaps between the algorithm's objective function value and the optimal objective function value. For instance, if the optimal tardiness is 0, then relative performance would be undefined (equal infinity).

Overall, we were able to solve all the instances in the SDS and 37 instances of the LDS to optimality.

Table 5
Performance of the IVNH and CPLEX on the LDS.

n	m	$[\underline{v}, \overline{v}]$	Release time	Due date	No. of found best known solutions		Avg. absolute performance		Avg. relative performance		Avg. objective value of best known solutions	Avg. abs. performance of the best known solution to the lower bound of CPLEX
					IVNH	CPLEX	IVNH	CPLEX	IVNH	CPLEX		
100	5	[10 30]	no	large	19	3	0.1	3.0	0.8%	65.9%	5.9	5.4
⌈	2	[10 30]	no	large	12	19	0.9	0.2	89.8%	30.0%	1.7	0.7
|	⌈	[0 100]	no	large	19	16	0.2	0.3	0.0%	0.0%	1,350.5	966.9
|	|	[0 20]	no	large	20	20	0.0	0.0	–*	–*	0.0	0.0
|	|	⌈	no	large	20	0	0.0	14.1	0.0%	122.7%	12.6	11.6
150	5	[10 30]	yes	large	20	0	0.0	12.0	0.0%	106.4%	12.9	11.8
|	|	|	no	medium	19	1	0.3	24.1	0.1%	6.6%	370.3	104.0
|	|	⌊	no	small	18	2	0.2	15.0	0.0%	1.4%	1,077.3	141.2
|	|	[20 40]	no	large	19	3	0.1	2.8	0.2%	11.4%	25.5	25.5
|	⌊	[30 50]	no	large	20	20	0.0	0.0	0.0%	0.0%	345.9	63.8
⌊	8	[10 30]	no	large	20	0	0.0	32.7	0.0%	106.3%	33.1	32.1
200	5	[10 30]	no	large	20	0	0.0	29.4	0.0%	159.1%	27.3	25.6

* Not defined if the best found objective value is 0.

Table 6
Performance of the IVNH and CPLEX on the SDS.

n	m	$[\underline{v}, \overline{v}]$	Release time	Due date	IVNS		
					Avg. CPU, sec.	No. of optimal solutions (out of 20)	Avg. absolute performance
	2	[10 30]	no	large	0.00	20	0.0
	⌈	[0 100]	no	large	0.39	20	0.0
	|	[0 20]	no	large	0.00	20	0.0
	|	⌈	no	large	0.01	20	0.0
30	5	[10 30]	yes	large	0.01	20	0.0
	|	|	no	medium	0.16	15	0.3
	|	⌊	no	small	0.16	16	0.2
	|	[20 40]	no	large	0.00	20	0.0
	⌊	[30 50]	no	large	0.20	20	0.0
	8	[10 30]	no	large	0.03	20	0.0

4.2. Computational experiments on the randomly generated data sets

In the following, we compare the performance of the IVNH and CPLEX in Section 4.2.1. Section 4.2.2 examines the main performance drivers of the IVNH and Section 4.2.3 compares the basic and the extended model formulations of CPLEX. We conclude with a comment on the idle times in the computed solutions (Section 4.2.4).

4.2.1. Comparative performance of the IVNH and CPLEX

Table 5 reports results of the IVNH and CPLEX for the LDS. We run CPLEX on the extended model formulation (1)-(5). The IVNH takes just about 1 minute on average and never more than 10 minutes per instance. And we limit the run time of CPLEX by *one hour per instance* to examine potential advantages of the IVNH under especially demanding conditions.

CPLEX solved just 37 out of 240 instances to optimality: 14 instances in setting (150, 2, [10 30], no, large), 20 instances in setting (150, 5, [0 20], no, large), and 3 instances in setting (150, 5, [30 50], no, large). The average absolute optimality gap, that is the difference between the CPLEX upper and the CPLEX lower bounds, remained quite large. In fact, CPLEX lower bounds were close to 0 in the majority of the settings (e.g., compare the last two columns in Table 5).

The IVNH required just 88 seconds per instance on average and performed at most 101 iterations for each instance. Table 5 illustrates that the IVNH clearly outperformed CPLEX in 9 out of 12 settings and found solutions of comparable quality as CPLEX in the remaining three settings (150, 2, [10 30], no, large), (150, 5, [0 20], no, large), and (150, 5, [30 50], no, large).

Note that we were not able to compare the results of the IVNH to optimal solutions of all the instances of the LDS. Indeed, CPLEX was not able to solve the LDS instances to optimality even within a several-hour run time per instance.

Therefore, in order to compare the results of the IVNH with optimal solutions, we run the algorithms on the SDS data set (see Table 6). We did not limit the run time of CPLEX. CPLEX required up to 70 minutes per instance. Overall, the IVNH found optimal solutions in 95.5% of cases (for 191 instances out of 200) in a fraction of a second. In a few (nine out of 200) cases, in which the IVNH was not able to find an optimal solution, the remaining gap to optimality (= absolute performance) was very low. It equaled 1 in eight instances and 2 in the ninth instance. Recall that the gap to optimality equals 1, for instance, if just one job has been assigned one bin later. Interestingly, all these nine instances belong to the settings with more restrictive – small and medium – due dates. The IVNH found optimal solutions for $\frac{15+16}{20+20} = 77.5\%$ of the instances in these two settings. A possible reason could be the following. Because of more restrictive due dates, optimal solutions are likely to contain many tardy jobs. As a consequence, we are more likely to be trapped in a local optimum, because many neighbors in the neighborhoods $\mathcal{N}_1, \mathcal{N}_2$, and $\mathcal{N}_3$ are not improving neighbors and will be rejected. So that the instances get harder for the IVNH and the relative advantage of enumeration-based procedures, such as CPLEX, becomes more pronounced.

We conclude that the IVNH performs very well. Overall, the relative advantage of the IVNH seems larger for a larger number of dimensions m and for a larger number of jobs n, see Fig. 6.

4.2.2. Performance drivers of the IVNH

In this section, we analyze main performance drivers of the IVNH. We start by examining the influence of the number of iterations, or so-called convergence behavior of the IVNH.

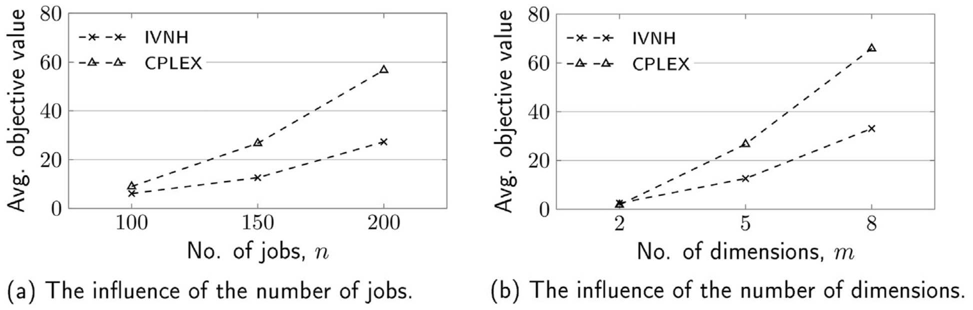


Fig. 6. Sensitivity analysis of the basic setting of the LDS instances – (150, 5, [10 30], no, large): Relative advantage of the IVNH over CPLEX with run time of 60 min.

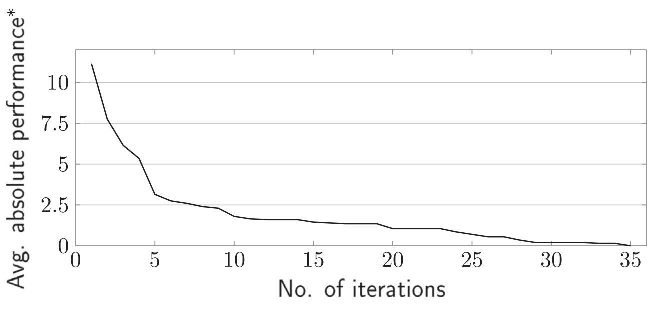


Fig. 7. Convergence analysis of the basic setting of the LDS instances – (150, 5, [10 30], no, large).
* The average absolute performance of the best incumbent solution found so far.

Recall that the following algorithmic steps constitute an iteration of the IVNH (cf. Section 3.4):

- construction of a new incumbent solution with procedures described in Section 3.1 or Section 3.3,
- local search with respect to neighborhoods $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$ (see Section 3.2).

Fig. 7 illustrates the convergence behavior for instances of the basic setting (150, 5, [10 30], no, large). For each iteration, the figure shows the average absolute performance of the best solution found until this iteration. As we see, the quality of the best found solution rapidly improves in the first five iterations and continuously ameliorates at a diminishing rate afterward. It is exactly the kind of behavior we would like to have in a good heuristic procedure, because (i) solutions of very good quality are found very fast and (ii) there are excellent chances to compute even better solutions if we relax the stopping criterion and perform more iterations, unless an optimal solution has been already found. Note that we found a similar behavior of the IVNH in other settings as well. For instance, in Table 7, we report the average number of iterations until the best solution was found that could not be improved in the $Glim = 20$ consequent iterations. Table 7 illustrates that, in concordance with intuition, the flattening-out of the average absolute performance of the IVNH occurs at later iterations for instances with a larger number of jobs or a larger number of dimensions.

Table 7
Average number of iterations until the best solution is found which is not improved in the $Glim = 20$ consequent iterations.

n	m	$[\underline{v}, \overline{v}]$	Release time	Due date	Avg. number of iterations
100	5	[10 30]	no	large	12.9
⌈	2	[10 30]	no	large	6.3
│	⌈	[0 100]	no	large	11.0
│	│	[0 20]	no	large	1.0
│	│	⌈	no	large	14.4
150	5	[10 30]	yes	large	22.1
│	│	│	no	medium	19.4
│	│	⌊	no	small	16.9
│	│	[20 40]	no	large	11.7
│	⌊	[30 50]	no	large	1.0
⌊	8	[10 30]	no	large	22.3
200	5	[10 30]	no	large	21.4

We also analyze the impact of different algorithmic elements on the performance of the IVNH in Table 8. In the initial algorithm of the IVNH, we disable local search procedures (see Section 3.2) one at a time. We also replace our customized diversification procedure (see Section 3.3) with a straightforward random generation of a new feasible solution by assigning jobs to the bins by the first-fit procedure (see Section 3.1) in a randomly generated order. We call the resulting algorithm as 'Random diversification' in Table 8.

According to the received results, local search with respect to neighborhood $\mathcal{N}_1$ has the largest impact on the IVNH performance in almost all the settings. Neighborhood $\mathcal{N}_3$ appears to be especially important in data settings with job sizes [10 30] and [0 100], i.e., data settings in which small jobs are present (the results on setting [0 20] are inconclusive because all the examined algorithms found solutions of very good quality). A possible explanation for this effect can be the following. If many small jobs are present, then feasible solutions are likely to have many jobs in each bin and each neighborhood move in $\mathcal{N}_3$ relocates many jobs at once. If the jobs are large in size, however, so that only very few (one, two, or three) jobs can be packed into one bin, then the 'bin moves' of neighborhood $\mathcal{N}_3$ are equivalent to a few (one, two, or three) subsequent swap moves of neighborhood $\mathcal{N}_1$ or a few subsequent reinsertion moves of neighborhood $\mathcal{N}_2$. In such settings, locally optimal solutions with respect to neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ are more likely to be locally optimal with respect to the neighborhood $\mathcal{N}_3$ as well and the relative impact of $\mathcal{N}_3$ in the IVNH performance gets smaller.

The impact of our diversification procedure is twofold. On the one hand, it improves the final solutions (see Table 8). On the other hand, it speeds up the IVNH-algorithm several times. In other words, our diversification procedure seems to construct promising new incumbent solutions, whereas the incumbent solutions generated by the random diversification are usually far from the local optima and need more steps of the local search (see also our discussion in Section 3.3).

4.2.3. *Comparison of the two integer programming model formulations*

In Table 9, we additionally illustrate performance of the strengthened model formulation (1)–(5) and the basic model formulation (1)–(4) on the SDS. In this computational experiment, we did not limit the run time of CPLEX. The strengthened model (1)–(5) reduced computational time of CPLEX significantly compared to the basic model. For example, in setting (30, 5, [30 50], no, large), the computational time was reduced by 69% on average and the number of nodes decreased to 0 for all 20 instances, i.e., all the instances were solved in the root node.

Table 8
Impact of the algorithmic elements of the IVNH on its performance.

n	m	$[\underline{v}, \overline{v}]$	Release time	Due date	Difference in the average absolute performance to that of the IVNH				CPU ratio to that of the IVNH
					Disabled $\mathcal{N}_1$	Disabled $\mathcal{N}_2$	Disabled $\mathcal{N}_3$	Random diversification	Random diversification
100	5	[10 30]	no	large	5.8	1.1	0.5	0.2	277%
⌈	2	[10 30]	no	large	4.0	1.0	0.7	0.1	313%
|	⌈	[0 100]	no	large	18.5	0.8	4.7	0.1	109%
|	|	[0 20]	no	large	0.0	0.0	0.0	0.1	–*
|	|	⌈	no	large	17.3	1.4	2.5	0.1	233%
150	5	[10 30]	yes	large	14.8	1.0	2.4	0.2	222%
|	|	|	no	medium	64.0	4.8	15.4	2.3	318%
|	|	⌊	no	small	48.2	2.4	15.8	4.5	262%
|	|	[20 40]	no	large	6.0	0.9	1.7	0.6	379%
|	⌊	[30 50]	no	large	0.0	0.0	0.0	0.0	769%
⌊	8	[10 30]	no	large	21.3	2.7	4.0	0.0	253%
200	5	[10 30]	no	large	31.7	1.9	3.6	0.0	172%

* Not defined if the CPU time of the IVNH is 0.

Table 9
CPLEX results for the two model formulations.

n	m	$[\underline{v}, \overline{v}]$	Release time	Due date	Model (1)–(4)		Model (1)–(5)	
					Avg. CPU, sec.	Avg. no. of nodes	Avg. CPU, sec.	Avg. no. of nodes
	2	[10 30]	no	large	0.06	0.0	0.06	0.0
	⌈	[0 100]	no	large	6.64	25,639.8	6.71	17,734.4
	|	[0 20]	no	large	0.06	0.0	0.06	0.0
	|	⌈	no	large	0.13	458.1	0.12	328.1
30	5	[10 30]	yes	large	0.23	2,426.4	0.12	353.8
	|	|	no	medium	395.88	1,871,376.8	115.84	658,962.1
	|	⌊	no	small	30.26	203,352.1	21.93	133,944.6
	|	[20 40]	no	large	0.07	0.0	0.08	0.0
	⌊	[30 50]	no	large	1.00	1,154.6	0.25	0.0
	8	[10 30]	no	large	15.92	90,686.8	2.68	15,731.2

4.2.4. Report on the idle times in the found solutions

Of course, the most important objective for the firms that produce customized products is meeting the promised delivery dates. Nevertheless, it is interesting to examine the dynamics of idle times in our computational experiments. Idle times are an important performance indicator of the cost-efficiency of the assembly line. Fig. 8(a) and (b) illustrate idle times for the basic setting of the LDS instances (150, 5, [10 30], no, large) and how it changes with the number of dimensions (Fig. 8(a)) and the job size (Fig. 8(b)). Interestingly, the average idle time remains moderate and does not exceed 20% for a large range of settings, which illustrates the efficiency of the multiple-piece-flow manufacturing policy.

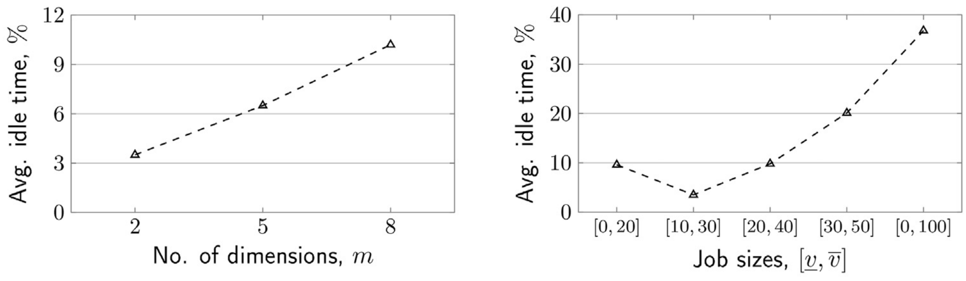
Fig. 8(b) illustrates that two opposite effects may influence idle times. On the one hand, the smaller the jobs are, the better we can combine them to fill bins to their capacity and the idle time decreases. The other effect is related to the peculiarity of the total tardiness function: any schedule, in which jobs are finished not later than their due dates, is equally good notwithstanding the amount of the idle time. In other words, if jobs are so small so that we can pack many jobs in one bin and finish them without delay, the idle time may increase as in the setting (150, 5, [0 20], no, large).

4.3. Simulation study: The IVNH as part of rolling-horizon planning

We have also examined our algorithm in a case study of the production of industrial cranes.

The aim of the current simulation study is to illustrate the suitability of the tardiness objective function in general and the designed heuristic IVNH in particular for the rolling-horizon planning framework used in the company. Therefore, we compare the IVNH to the status quo planning rule and perform further analysis of its performance in Section 4.3.2. In Section 4.3.3, we compare the tardiness objective function to another widespread objective function – minimization of the total idle time. We describe the simulation framework in Section 4.3.1.

Based on the results in Section 4.2, we do not use CPLEX in our simulation experiments for the following two reasons:



(a) The influence of the number of dimensions (b) The influence of job sizes

Fig. 8. Sensitivity analysis of the basic setting of the LDS instances– (150, 5, [10 30], no, large): The average idle time.

- The resulting problem instances that have to be solved within the simulation framework are too large to be solved by CPLEX in a reasonable amount of time. Observe that in the rolling-horizon planning framework, an m-BiPacS instance have to be solved at each replanning step.
- If we set some acceptable run time limit, CPLEX performs worse than the IVNH.

We also conducted some selective preliminary testing, which reconfirmed the reasons stated above.

4.3.1. Data generation

To simulate the actual production planning process, we set up a rolling-horizon planning framework: The simulation runs for 15 days and the replanning is performed in the beginning of each day.

The product consists of two large workpieces – a pillar and a jib –, each of them is processed in a separate assembly line consisting of four and five stations, respectively (see Fig. 2). Workpieces should be cleansed, trimmed to the desired size, drilled, and welded. Each process requires a specialized machine and processing times highly depend on the customer orders, such as on the length and diameter of the pillars or jibs and on the ordered fixtures. Afterward, the two workpieces are joined and processed together at the final station. So that there are $4+5+1=10$ dimensions in total. A bundle of workpieces (a bin of jobs) enters each station in the beginning of the day and leaves the station in the beginning of the next day. There is a buffer before the final station to compensate for different lengths of the two assembly lines. The work on each station is organized in shifts, and bottleneck stations work several shifts per day, whereas the rest stations work just one shift or a half shift a day. We have taken these different shift schedules into account by an appropriate scaling of the dimensions' capacities c and job sizes v_{ji}.

We have setup the arrival process of the jobs to mimic the actual company data, for instance, in the expected number of the arriving jobs each day and in the distribution of different crane models. Following the practice data, we also start the simulation with a four-day backlog of jobs, i.e., the jobs arrived in the last five days (including the current day) are available for immediate processing. In the beginning of the next day, the set of available jobs reduces by the bin load whose manufacture has already started and increases by the newly arrived jobs; a new plan is made for the now available jobs. Because we simulate the production process for 15 days, we perform 15 replannings in total. At each replanning step of the rolling-horizon planning framework, all the available jobs are scheduled. Because the simulated system is in the steady state, the number of the scheduled jobs roughly corresponds to the work amount for the next five days. Thereby, in the default setting (in contrast to the setting with full information), the planner only knows the characteristics of the already arrived jobs. In each simulation setting, we generate 100 simulation instances (runs) and report the average results obtained in these runs. We have also carefully checked the simulated data for possible warm-up and phasing-out effects. As a consequence, we have truncated the last four bins in the simulated data in the idle time experiment (Fig. 10(a) in Section 4.3.3). Indeed, because of the phasing-out effect, less jobs were available for planning and four last bins had moderately larger idle times.

We formulate several simulation settings to examine different promised delivery times u: $d_j = r_j + u$, where r_j is the arrival time of the job. The initial level of u, denoted as u_0, was calibrated to the current policy of the company – the time between the final specification of the job (i.e., crane characteristics) and the latest acceptable start of production given the promised delivery time. At the request of the company management, we also examine the

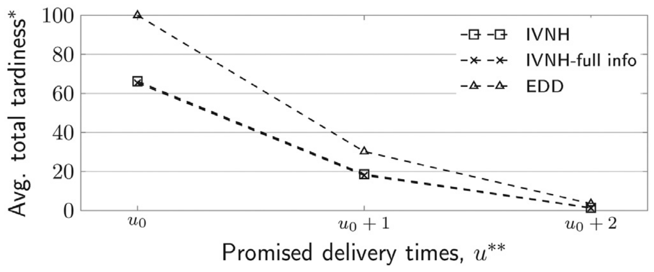


Fig. 9. Results of the simulation study.
*The objective value computed by the EDD for $u = u_0$ is set to 100.
**The values of u are anonymized, the minimal value is denoted as u_0.

values of $u = u_0 + 1$ and $u = u_0 + 2$ in our experiments, which correspond to later promised delivery times.

4.3.2. Tardiness objective function in the rolling-horizon planning framework

In this section, we examine the suitability of the set up optimization problem with the tardiness objective function for the rolling-horizon planning framework used in the company. For instance, to investigate the improvement of the status quo, we compare the results of the IVNH to the earliest due date (EDD) heuristic (see Section 3.1) which is common in practice and roughly describes the currently used planning routine in the company. Recall that in the rolling-horizon planning framework, the information is incomplete: The planner only knows the characteristics of the jobs that have already arrived and knows nothing about the jobs that are about to enter the system in the future. Therefore, we also perform the *competitive analysis*, i.e., we compare the results of the IVNH in this incomplete-information framework to the results of the IVNH in the full-information case (designated as the *IVNH-full info*). The latter schedules jobs in the beginning of day 1 with a full information about all the future jobs and their release times (i.e., times of arrival).

Fig. 9 illustrates the average objective value (tardiness) in the solutions found by the IVNH, EDD, and the IVNH-full info. To anonymize the data, we set the average delay per job for the EDD at initial value u_0 to 100.

As expected, the IVNH significantly outperforms the EDD, especially at shorter promised delivery times. If the promised delivery times are large and set at $u = u_0 + 2$, the jobs can be easily manufactured without tardiness and the advantage of the IVNH over the EDD vanishes out.

Interestingly, the value of full information, which is the difference in the objective value computed with the IVNH-full info compared to the results of the IVNH, is modest. The actual gaps between both results are 0.8, 0.6, and 0.1 for u_0, u_0+1 and u_0+2 respectively. We conclude that the tardiness objective function performs very well in the rolling-horizon procedure for the studied company.

Overall, the total tardiness in Fig. 9 is nonlinear in u: it increases sharply if the promised delivery times u become short. Indeed, in case of less restrictive promised delivery times, jobs may be combined more freely into bins, so that more jobs can be produced in the given time period. We have also observed that at low values of u large urgent jobs are pushed back into the later bins in order to fill earlier bins with many smaller jobs.

4.3.3. Comparison of the tardiness and the idle-time objective functions

In this section, we compare the tardiness objective function to the idle time objective function in the rolling-horizon planning framework. The minimization of the average idle time per station (i.e., per dimension) leads to lower manufacturing costs per product unit. This is a widespread objective function in the assembly

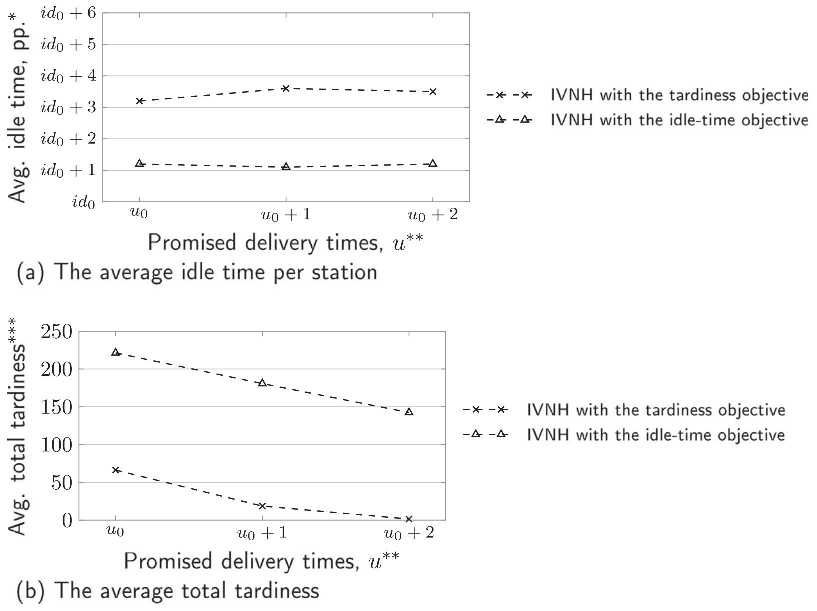


Fig. 10. Comparison of two different objective functions in the rolling-horizon planning framework: the minimization of the total tardiness and the minimization of the total idle time.
*The data is anonymized, the scale starts with some initial idle time level id_0.
**The values of u are anonymized, the minimal value is denoted as u_0.
***The objective value computed by the EDD for $u = u_0$ is set to 100.

line balancing and bin packing literature. Observe that because the sizes of each bin dimension and the sizes of the jobs are given, the minimization of the average idle time per station is equivalent to the minimization of the total idle time and to the minimization of the number of the bins.

The adaptation of the IVNH algorithm from Section 3 to the idle-time objective function is straightforward and simply amounted to using the new objective function in the respective algorithmic routines, such as selection of an improving neighbor, update of the best found solution etc. Observe, however, that the local search with respect to neighborhood $\mathcal{N}_3$ turns obsolete, since we cannot improve the average idle time by resequencing the bin loads.

Fig. 10(a) compares the average idle time per station in case the rolling-horizon planning uses the IVNH with the tardiness and with the idle-time objective functions, respectively. As expected, the average idle time in the latter case is lower. But the degree of improvement, which equals about 2.3 percentage points, is considered to be very moderate by the company management.

The average total tardiness of the manufactured jobs is presented in Fig. 10(b). As we see, the average total tardiness is extremely high in case the rolling-horizon planning uses the IVNH with the idle-time objective function, for instance, it is about 2.3 times larger compared to the status-quo rule EDD and $u = u_0$. Indeed, the solution algorithm often pushes some urgent jobs to the ever later bins. Interestingly, the average tardiness remains very high even if the promised delivery times are set large, at $u = u_0 + 2$.

We conclude that the idle time objective function is not suitable for the rolling-horizon planning framework.

5. Conclusion

This paper studies product sequencing in paced multiple-piece-flow assembly lines that are used, for instance, by medium-sized companies in manufacturing large pieces of customized equipment. We set up a mixed-integer model and analyze the relation of the problem, which we call m-vector bin packing and sequencing problem (m-BiPacS), to classical optimization problems in the literature. We design an iterative variable neighborhood search metaheuristic, the IVNH, to address large problem instances of the m-BiPacS. The IVNH utilizes several innovative local search procedures that examine many distinct good-quality solutions with maximally packed bins in a short time. In our computational experiments based on well-diversified randomly generated data sets, the IVNH clearly outperformed the off-the-shelf optimization tool IBM ILOG CPLEX: It found better solutions and had much shorter run times. For smaller problem instances with known optimal solutions, the IVNH was able to find an optimal solution in a fraction of a second in almost all cases. The gap to optimality in the remaining cases was negligible. An extended simulation based on real-world data confirms suitability of the IVNH and the tardiness objective function for the rolling-horizon planning framework in practice.

Future studies may develop exact solution methods that can solve instances of practice-relevant size in acceptably short run times. Research is needed on related planning problems, such as assembly line balancing and committing order due dates in multiple-piece-assembly lines. In particular, our simulation illustrates the importance of accurately estimated promised delivery dates. Promised delivery dates that are too restrictive prohibit a favorable combination of jobs (workpieces) into bins (lots that are launched simultaneously) and manufacturing cost may increase significantly. Overall, companies need the support of academia in developing further organizational concepts and planning tools that help to reduce manufacturing costs in the production of highly customized products. For instance, future field and case studies may investigate whether the performance of multiple-piece-flow assembly lines can be further improved by flexible deployment of machines (robots) and workers and more flexible flow of workpieces.

Appendix A

In this appendix, we explain the relation of $\mathcal{N}_1$ to the conventional two-job exchange neighborhood.

Recall that we get neighbor s'_c of some solution s in the conventional two-job exchange neighborhood by swapping two jobs belonging to two different bins such that the resulting solution

Table 10
Comparison of the $\mathcal{N}_1$ neighborhood and the conventional two-job exchange neighborhood.

n	m	$[\underline{v}, \bar{v}]$	Release time	Due date	$\mathcal{N}_1$			Conventional two-job exchange neighborhood			
					Avg. total No. of neighbors	Avg. No. of distinct neighbors	Avg. absolute performance of the best neighbor	Avg. total No. of neighbors	Avg. No. of distinct neighbors*	Avg. No. max. loaded neighbors**	Avg. absolute performance of the best neighbor
100	5	[10 30]	no	large	4,950.0	4,703.1	145.4	2,186.1	2,186.1	2,180.2	161.8
⌈	2	[10 30]	no	large	11,175.0	10,739.0	253.0	3,997.0	3,997.0	3,988.0	282.6
│	⌈	[0 100]	no	large	11,175.0	10,743.3	2,209.4	4,759.2	4,759.2	4,759.2	2,280.8
│	│	[0 20]	no	large	11,175.0	9,940.7	89.3	1,008.9	1,008.9	1,000.8	109.6
│	│	⌈	no	large	11,175.0	10,788.5	340.5	4,630.1	4,630.1	4,620.8	371.7
150	5	[10 30]	yes	large	11,175.0	10,286.5	325.4	3,628.0	3,628.0	3,620.7	365.3
│	│	│	no	medium	11,175.0	10,785.3	868.2	4,432.5	4,432.5	4,424.8	924.2
│	│	⌊	no	small	11,175.0	10,782.9	1,551.2	4,285.7	4,285.7	4,279.9	1,610.4
│	│	[20 40]	no	large	11,175.0	10,846.6	406.9	2,439.0	2,439.0	2,438.4	421.5
│	⌊	[30 50]	no	large	11,175.0	11,100.0	933.5	11,100.0	11,100.0	11,100.0	933.5
⌊	8	[10 30]	no	large	11,175.0	10,816.1	394.3	5,077.8	5,077.8	5,072.2	421.9
200	5	[10 30]	no	large	19,900.0	19,372.4	601.9	8,164.5	8,164.5	8,153.5	657.4

* The neighbor solutions with *not* maximally packed bins are also included.
** After subsequent sequential shift of jobs.

is feasible. We say that a *subsequent sequential shift of jobs* is performed to solution s'_c, when we shift the jobs belonging to some bin to the earliest possible bins to which they can be assigned; we apply this procedure to all the bins in the increasing order of the bin number. We call the resulting solution s''_c. For example, consider the instance in Table 3 and $s = \{1,2\}\{3,4\}\{5,6\}\{7,8\}$. Feasible solution $s'_c = \{1,\mathbf{8}\}\{3,4\}\{5,6\}\{7,\mathbf{2}\}$ is a neighbor solution of s with respect to the conventional two-job exchange neighborhood. After the subsequent sequential shift of jobs, we receive $s''_c = \{1,\mathbf{5},8\}\{3,4\}\{6,\mathbf{7}\}\{2\}$.

Lemma 2. *Neighborhood $\mathcal{N}_1$ of some feasible solution s contains all the neighbor solutions of the conventional two-job exchange neighborhood of s that are improved by subsequent sequential shift of jobs to receive maximally packed bins. In other words, for each neighbor solution s'_c of s with respect to the conventional two-job exchange neighborhood, there exists a feasible solution s''_c that is part of neighborhood $\mathcal{N}_1(s)$ and that can be constructed from s'_c with the subsequent sequential shift of jobs.*

Proof. The proof follows straightforwardly from the definition of $\mathcal{N}_1$. Indeed, to construct $s'_{\mathcal{N}_1} \in \mathcal{N}_1(s)$, we exchange job j with some job j' that belongs to another bin in s in steps 1 and 2 in Fig. 3(ii). The subsequent application of the first-fit procedure can be interpreted as the subsequent sequential shift of jobs. □

Appendix B

In this appendix, we perform a computational experiment on the LDS instances (see Section 4.1) to compare the number of neighbors in $\mathcal{N}_1$ and the conventional two-job exchange neighborhood.

We construct an initial solution with the PRBM heuristic described in Section 3.1, afterward we enumerate all the neighbors of this solution in the respective neighborhood. Observe that some neighbors in $\mathcal{N}_1$ may represent essentially identical solutions, therefore, we also compute the number of distinct neighbors. Table 10 summarizes the results. Overall, neighborhood $\mathcal{N}_1$ contained 242.6% more neighbors and 221.8% more distinct neighbors on average than the conventional two-job exchange neighborhood. Recall that neighbors in the conventional neighborhood may contain *not* maximally packed bins. After we applied a subsequent sequential shift of jobs to the neighbors in the conventional neighborhood (cf. Appendix A), the number of distinct solutions reduced even further. As a consequence, average absolute performance of the best neighbor was 8.5% worse in the conventional neighborhood, even after the subsequent sequential shift of jobs had been applied.

This computational experiment confirmed that $\mathcal{N}_1$ tend to contain more neighbors and, therefore, its best improving neighbor may have a better objective function value than the conventional two-job exchange neighborhood.

References

[1] Aloulou MA, Bouzaiene A, Dridi N, Vanderpooten D. A bicriteria two-machine flow-shop serial-batching scheduling problem with bounded batch size. J Sched 2014;17:17–29.
[2] Alves C, de Carvalho JV, Clautiaux F, Rietz J. Multidimensional dual-feasible functions and fast lower bounds for the vector packing problem. Eur J Oper Res 2014;233(1):43–63.
[3] Battaïa O, Dolgui A. A taxonomy of line balancing problems and their solution approaches. Int J Prod Econ 2013;142(2):259–77.
[4] Battaïa O, Otto A, Sgarbossa F, Pesch E. Future trends in management and operation of assembly systems: from customized assembly systems to cyber-physical systems. Omega 2018;78:1–5.
[5] Bautista J, Alfaro-Pozo R, Batalla-García C. GRASP For sequencing mixed models in an assembly line with work overload, useless time and production regularity. Prog Artif Intell 2016;5(1):27–33.
[6] Baybars İ. A survey of exact algorithms for the simple assembly line balancing problem. Manag Sci 1986;32(8):909–32.
[7] Becker C, Scholl A. A survey on problems and methods in generalized assembly line balancing. Eur J Oper Res 2006;168(3):694–715.
[8] Bock S, Rosenberg O, Brackel TV. Controlling mixed-model assembly lines in real-time by using distributed systems. Eur J Oper Res 2006;168:880–904.
[9] Boysen N, Fliedner M, Scholl A. A classification of assembly line balancing problems. Eur J Oper Res 2007;183(2):674–93.
[10] Boysen N, Fliedner M, Scholl A. Assembly line balancing: which model to use when? Int J Prod Econ 2008;111(2):509–28.
[11] Boysen N, Fliedner M, Scholl A. Production planning of mixed-model assembly lines: overview and extensions. Prod Plan Control 2009a;20(5):455–71.
[12] Boysen N, Fliedner M, Scholl A. Sequencing mixed-model assembly lines: survey, classification and model critique. Eur J Oper Res 2009b;192(2):349–73.
[13] Brucker P. Scheduling algorithms. Berlin, Heidelberg: Springer; 2007.
[14] Brucker P, Gladky A, Hoogeveen H, Kovalyov MY, Potts C, Tautenhahn T, et al. Scheduling a batching machine. J. Schedul. 1998;1:31–54.
[15] Bukchin J. A comparative study of performance measures for throughput of a mixed model assembly line in a JIT environment. Int J Prod Res 1998;36:2669–85.
[16] Buxey GM, Slack ND, Wild R. Production flow line system design – a review. AIIE Trans. 1973;5(1):37–48.
[17] Cabo M, González-Velarde JL, Possani E, Solís YÁR. Bi-objective scheduling on a restricted batching machine. Comput. Oper. Res. 2018;100:201–10.
[18] Cabo M, Possani E, Potts CN, Song X. Split-merge: using exponential neighborhood search for scheduling a batching machine. Comput. Oper. Res. 2015;63:125–35.
[19] Camati RS, Calsavara A, Jr LL. Solving the virtual machine placement problem as a multiple multidimensional knapsack problem. In: Gyires T, Westphall CB, Kálmán G, editors. Proceedings of the thirteenth international conference on networks, ICN; 2014. p. 253–60.
[20] Caprara A, Kellerer H, Pferschy U. Approximation schemes for ordered vector packing problems. Nav Res Logist 2003;50(1):58–69.

[21] Caprara A, Toth P. Lower bounds and algorithms for the 2-dimensional vector packing problem. Discret Appl Math 2001;111(3):231–62.
[22] Chang SY, Hwang H-C, Park S. A two-dimensional vector packing model for the efficient use of coil cassettes. Comput Oper Res 2005;32(8):2051–8.
[23] Cheng TCE, Kovalyov MY. Single machine batch scheduling with sequential job processing. AIIE Trans 2001;33:413–20.
[24] Christensen HI, Khan A, Pokutta S, Tetali P. Multidimensional Bin Packing and Other Related Problems: A Survey; 2016. http://people.math.gatech.edu/~tetali/PUBLIS/CKPT.pdf.
[25] Copil K, Wörbelauer M, Meyr H, Tempelmeier H. Simultaneous lotsizing and scheduling problems: a classification and review of models. OR Spectr 2017;39(1):1–64.
[26] Cortez PM, Costa AM. Sequencing mixed-model assembly lines operating with a heterogeneous workforce. Int J Prod Res 2015;53(11):3419–32.
[27] Davis SM. Future perfect. Reading, MA: Addison-Wesley; 1987.
[28] Dolgui A, Guschinsky N, Levin G. A special case of transfer lines balancing by graph approach. Eur J Oper Res 2006;168:732–46.
[29] Dörmer J, Günther HO, Gujjula R. Master production scheduling and sequencing at mixed-model assembly lines in the automotive industry. Flex Serv Manuf J 2015;27(1):1–29.
[30] Epstein L, Levin A. Bin packing with general cost structures. Math Program 2012;132:355–91.
[31] Erel E, Sarin SC. A survey of the assembly line balancing procedures. Prod Plan Control 1998;9:414–34.
[32] Faccio M, Gamberi M, Bortolini M. Hierarchical approach for paced mixed–model assembly line balancing and sequencing with jolly operators. Int J Prod Res 2016;54(3):761–77.
[33] Fleischmann B. The discrete lot-sizing and scheduling problem. Eur J Oper Res 1990;44(3):337–48.
[34] Fleischmann B, Meyr H. The general lotsizing and scheduling problem. OR Spektrum 1997;19(1).
[35] Freville A. The multidimensional 0–1 knapsack problem: an overview. Eur J Oper Res 2004;155:1–21.
[36] Gabay M, Zaourar S. Vector bin packing with heterogeneous bins: application to the machine reassignment problem. Ann Oper Res 2016;242(1):161–94.
[37] Garey M, Graham R, Johnson D, Yao A. Resource constrained scheduling as generalized bin packing. J Comb Theory: Ser A 1976;21:257–98.
[38] Gavish B, Pirkul H. Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. Math Program 1965;31:78–105.
[39] Ghosh S, Gagnon RJ. A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. Int J Prod Res 1989;27(4):637–70.
[40] Glover F. Heuristics for integer programming using surrogate constraints. Decis Sci 1977;8(1):156–66.
[41] Haase K. Capacitated lot-sizing with sequence dependent setup costs. OR Spektrum 1996;18(1):51–9.
[42] Heßler K, Gschwind T, Irnich S. Stabilized branch-and-price algorithms for vector packing problems. Discussion paper Series. Johannes Gutenberg University of Mainz; 2017. 9th of August.
[43] Hitchcock FL. The distribution of a product from several sources to numerous localities. J Math Phys 1941;20:224–30.
[44] Hu Q, Wei L, Lim A. The two-dimensional vector packing problem with general costs. Omega (Westport) 2018;74:59–69.
[45] Jaber MY, editor. Learning curves: theory, models, and applications. CRC Press; 2016.
[46] Jackson JR. A computing procedure for a line balancing problem. Manag Sci 1956;2:261–71.
[47] Jonker R, Volgenant A. A shortest augmenting path algorithm for dense and sparse linear assignment problems. Computing 1987;38(4):325–40.
[48] Kellerer H, Pferschy U, Pisinger D. Knapsack problems. Berlin: Springer; 2004.
[49] Kleinschmidt P, Schannath H. A strongly polynomial algorithm for the transportation problem. Math Program 1995;68:1–13.
[50] Laabadi S, Naimi M, Amri HE, Achchab B. The 0/1 multidimensional knapsack problem and its variants: a survey of practical models and heuristic approaches. Am J Oper Res 2018;8:295–439.
[51] Lapierre SD, Ruiz A, Soriano P. Balancing assembly lines with tabu search. Eur J Oper Res 2006;168:826–37.
[52] Leung JY-T, Li C-L. An asymptotic approximation scheme for the concave cost bin packing problem. Eur J Oper Res 2008;191(2):582–6.
[53] Li C-L, Chen Z-L. Bin-packing problem with concave costs of bin utilization. Nav Res Logist 2006;53(4):298–308.
[54] Martello S, Toth P. Knapsack problems: algorithms and computer implementations. New York: Wiley; 1990.
[55] Mayrhofer W, März L, Sihn W. Simulation-based optimization of personnel assignment planning in sequenced commercial vehicle assembly: a software tool for iterative improvement. J Manuf Syst 2013;32:423–8.
[56] Mosadegh H, Ghomi SF, Süer GA. A control theoretical modelling for velocity tuning of the conveyor belt in a dynamic mixed-model assembly line. Int J Prod Res 2017;55(24):7473–95.
[57] Mönch L, Fowler J, Dauzére-Pèrés S, Mason SJ, Rose O. A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. J Sched 2011;14(6):583–99.
[58] Ng VTY, Chan B, Shun LLY, Tsang R. Quality service assignments for role-based web services. In: Proceedings of the IEEE international conference on systems, man and cybernetics; 2008. p. 2219–24.
[59] Nof SY, Wilhelm W, Warnecke H. Industrial assembly. Springer–Science+Business Media, B.V.; 1997.
[60] Otto A, Battaïa O. Reducing physical ergonomic risks at assembly lines by line balancing and job rotation: a survey. Comput Ind Eng 2017;111:467–80.
[61] Otto A, Otto C. How to design effective priority rules: example of simple assembly line balancing. Comput Ind Eng 2014a;69:43–52.
[62] Otto A, Scholl A. Incorporating ergonomic risks into assembly line balancing. Eur J Oper Res 2011;212:277–86.
[63] Otto C, Otto A. Extending assembly line balancing problem by incorporating learning effects. Int J Prod Res 2014b;52:7193–208.
[64] Pine BJ. Mass customization: the new frontier in business competition. Boston, MA: Harvard Business School; 1993.
[65] Potts C, Van Wassenhove L. Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. J Oper Res Soc 1992;43:395–406.
[66] Potts CN, Kovalyov MY. Scheduling with batching: a review. Eur J Oper Res 2000a;120:228–49.
[67] Puchinger J, Raidl GR, Pferschy U. The multidimensional knapsack problem: structure and algorithms. INFORMS J Comput 2010;22:250–65.
[68] Rekiek B, Delchambre A. Assembly line design: the balancing of mixed-Model hybrid assembly lines with genetic algorithms. London: Springer; 2006.
[69] Sarker BR, Pan H. Designing a mixed-model, open-station assembly line using mixed-integer programming. J Oper Res Soc 2001;52:545–58.
[70] Scholl A, Becker C. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. Eur J Oper Res 2006;168(3):666–93.
[71] Simaria A, de Sá MZ, Vilarinho P. Meeting demand variation using flexible u-shaped assembly lines. Int J Prod Res 2009;47:3937–55.
[72] Song Y, Zhang C, Fang Y. Multiple multidimensional knapsack problem and its applications in cognitive radio networks. In: Proceedings of the MILCOM: military communications conference 2008. Institute of Electrical and Electronics Engineers; 2008. p. 3963–9.
[73] Varnamkhasti MJ. Overview of the algorithms for solving the multidimensional knapsack problems. Adv Stud Biol 2012;4:37–47.
[74] Wang H-M. Solving single batch-processing machine problems using an iterated heuristic. Int J Prod Res 2011;49(14):4245–61.
[75] Webster S, Baker K. Scheduling groups of jobs on a single machine. Oper Res 1995;43:692–703.
[76] Wei L, Zhu W, Lim A. A goal-driven prototype column generation strategy for the multiple container loading cost minimization problem. Eur J Oper Res 2015;241:39–49.
[77] Winston WL. Operation research: applications and algorithms. Brooks/Cole: Belmont; 2004.
[78] Yuan JJ, Lin YX, Cheng TCE, Ng CT. Single machine serial-batching scheduling problem with a common batch size to minimize total weighted completion time. Int J Prod Econ 2007;105:402–6.