Operations Research Perspectives 15 (2025) 100362

Contents lists available at ScienceDirect

Operations Research Perspectives

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

Customer order scheduling in a permutation flow shop environment

Julius Hoffmann [a,b,*], Janis S. Neufeld [b,c], Udo Buscher [b]

[a] Institute for Operations Research, Discrete Optimization and Logistics, Karlsruhe Institute of Technology, Kaiserstr. 89, Karlsruhe, 76133, Germany
[b] Faculty of Business and Economics, Chair for Industrial Management, TU Dresden, Helmholtzstr. 10, Dresden, 01069, Germany
[c] Chair of Operations Management, Otto-von-Guericke-Universität Magdeburg, Universitätsplatz 2, Magdeburg, 39106, Germany

ARTICLE INFO

Dataset link: https://github.com/Julius2627/COSP_Flow_Shop

Keywords:
Machine scheduling
Customer order scheduling
Iterated greedy algorithm
Metaheuristics
Flow shop

ABSTRACT

Various recent scheduling literature has studied the so called customer order scheduling problem. In this class of scheduling problems, there are multiple customer orders, and each of them consists of several jobs. The order finishes and is ready to be shipped when the last job of the order finishes. In this paper, we consider the customer order scheduling problem in a permutation flow shop environment with m machines. There are n orders and each order has o jobs. The objective is to minimize the total completion time of the orders. We present multiple problem properties and a MINLP formulation of the problem. Furthermore, four heuristics which follow the Iterated Greedy Algorithm are developed. In a computational experiment, we evaluated the four heuristics on their practicability. They showed good results in short calculation time when compared with the MINLP solution from a solver. Afterwards, we compared the four heuristics with each other for different problem sizes.

1. Introduction

In most real-world cases, a customer orders not one but multiple desired products. This applies to both the B2B and B2C sectors. To save on transportation costs, the manufacturer collects the products of a customer order and sends them all at once rather than sending each product individually. For the case that the customer needs all products together for further processing, this also comes along with lower warehousing costs for the customer, as prematurely arrived products must be stored until the further products of the order arrive. However, the described aspect is disregarded in classic scheduling problems, where each job is treated individually. Consequently, recent papers examine the so-called customer order scheduling problem (COSP). In this class of scheduling problems, each job to schedule belongs to a predefined customer order. A customer order finishes when each of the jobs has finished [1].

The principle of the COSP is not limited to the production context but can be transferred to other disciplines, e.g., to computational work (see [2] for explanation) or mobile communication. In the latter application area, a message consists of multiple packages which can be sent independently via the communication network to the receiver. The message is sent completely when all packages have arrived at the receiver.

The COSP has been studied for different problem settings and objective functions. A notation for the so-called assembly scheduling problem, for which the COSP is a special case, is presented by Framinan et al. [3]. In general, assembly scheduling problems consider a machine environment where at some point in the production process, all parts of a product are assembled. Depending on the concrete problem configuration, the product is finished after the assembly or needs further processing steps. The notation $\alpha_1 \rightarrow \alpha_2$ in the machine field of the classical three-field scheduling notation indicates that an assembly scheduling problem is considered. According to this notation, α_1 is the machine environment prior the assembly and α_2 the following machine environment. The COSP is a subclass of the assembly scheduling problem, where no further production step occurs after the assembly, i.e., $\alpha_2 = 0$ and hence, is represented as $\alpha_1 \rightarrow 0$ in the machine field of the classical three-field scheduling notation, i.e., $\alpha_1 \rightarrow 0|\beta|\gamma$. We are adopting the notation in this paper. Even though, the flow shop machine environment is prominent in scheduling research, literature about this machine environment for the COSP is sparse. To the best of our knowledge, this paper is the first that addresses the problem $Fm \rightarrow 0|prmu|\sum C_i$, i.e., minimizing the total completion time in a m-machine permutation flow shop environment for the COSP.

In recent years, the iterated greedy algorithm (IGA) has been used in various papers to solve different kinds of flow shop scheduling problems. It showed promising results when applied to a permutation based solution representation, which is common for the flow shop machine environment [4]. However, as the length of the job permutation

* Corresponding author at: Institute for Operations Research, Discrete Optimization and Logistics, Karlsruhe Institute of Technology, Kaiserstr. 89, Karlsruhe, 76133, Germany.
E-mail addresses: julius.hoffmann@kit.edu (J. Hoffmann), janis.neufeld@ovgu.de (J.S. Neufeld), udo.buscher@tu-dresden.de (U. Buscher).

https://doi.org/10.1016/j.orp.2025.100362
Received 4 July 2025; Received in revised form 30 October 2025; Accepted 31 October 2025
Available online 3 November 2025

increases, disadvantages arise with the IGA, as the objective function of (partial) solutions has to be calculated frequently during each iteration. This occurs in particular at the COSP in a flow shop environment, since the length of the job permutation depends on the number of orders and the number of jobs per order.

In this paper we address this issue and investigate the problem $Fm \rightarrow 0|prmu|\sum C_i$. This includes the presentation of four variants of the IGA which solve this problem efficiently. To be more specific, the contributions of this study are the following:

- We identify and point out a new problem configuration of the COSP, and derive properties of the problem which are mandatory for the program formulation and the IGAs.
- We formulate a MINLP of the problem configuration.
- We develop four IGAs for the problem which take the special problem structure into account. One is modifying the job permutation as a whole, while the others modify the job permutation of each order separately and subsequently merge these smaller permutations.
- The presented algorithms are evaluated in computational experiments.
- Based on the experimental results, we give suggestions for appropriate solution strategies for different problem sizes of $Fm \rightarrow 0|prmu|\sum C_i$.

The remainder of this article is structured as follows. Section 2 gives an overview about literature related to the COSP, the flow shop machine environment, and the IGA and its applications. In Section 3 we define the studied problem configuration and the notation which is used in the paper. Furthermore, we present the relevant problem properties. The developed IGAs are described in Section 4 and evaluated in the computational experiment of Section 5. Section 6 concludes the results of the study and notes possibilities of future research.

2. Literature review

In this section we present related literature which addresses the COSP in general, the COSP in machine settings which are similar to the machine environment considered in this paper, and the application of the IGA for different scheduling problems.

One of the earliest publications about the COSP is the paper from Wagneur and Sriskandarajah [5]. They studied the COSP in the dedicated machine environment, where each order has one job on each machine. However, the problem is not called COSP in the mentioned paper, but *open shop with jobs overlap*. This points out that the COSP in the dedicated machine environment can be considered as an open shop, where the restriction that a job cannot be processed simultaneously by the machines is removed. Note that an order corresponds to a job in this open shop environment and a job to a task.

Wagneur and Sriskandarajah [5] studied the complexity of some regular performance measures in the dedicated machine environment and showed that the makespan and the maximum lateness can be solved in polynomial time in this machine environment. Furthermore, they stated proofs that the problems of minimizing the total tardiness, the number of late jobs, and the total completion time are NP-hard, even if there are only two machines. However, as shown in [6], the proof for the last mentioned problem was not correct. The complexity of the problem remained open until Roemer proved that $DP2 \rightarrow 0||\sum C_i$ is indeed NP-hard [7].

The dedicated machine environment was further investigated by Leung et al. [8]. They first showed an interesting property of this machine environment, namely that there exists always an optimal schedule for $DPm \rightarrow 0||\sum f_i\left(C_i\right)$, where $f_i\left(C_i\right)$ is increasing in C_i for each i, where each order is processed in the same sequence on each machine. This property significantly facilitated the handling of problem solving for objective functions such as $\sum C_i$, $\sum T_i$, and $\sum U_i$ as a schedule could be represented by a sequence of order IDs. The authors used this insight to describe six heuristics, including the Shortest Total Processing Time first heuristic, the Earliest Completion Time first heuristic and a Tabu Search heuristic, to solve the problem $DPm \rightarrow 0||\sum C_i$.

As it is one of the most prominent objective functions in the scheduling literature, the total completion time in connection with the COSP was also examined in other papers. Yang and Posner [9] investigated the total completion time of the COSP in a parallel machine environment with identical machines. They stated that for the problem $Pm \rightarrow 0||\sum C_i$ there exists an optimal schedule, where the jobs of an order which are scheduled on the same machine, are processed consecutively. Furthermore, they presented two heuristics for $P2 \rightarrow 0||\sum C_i$ and one for $Pm \rightarrow 0||\sum C_i$. The unrelated parallel machine environment with the product type splitting property was examined by Xu et al. [10]. After establishing some optimality properties and a programming formulation, the authors presented three heuristics for this problem. More recently, the dedicated machine environment has once again been the subject of research in [1,11], where different heuristic approaches were studied.

There are various further problem configurations of the COSP studied in the literature, e.g., $DPm \rightarrow 0||\sum T_i$ in [12–14], $DPm \rightarrow 0|r_i|\sum w_iU_i$ in [15], $DPm \rightarrow 0|ST_{sd}|C_{max}$ in [16], $DPm \rightarrow 0|ST_{sd}|\sum C_i$ in [17], and $Pm \rightarrow 0|batch|\sum w_iC_i$ in [18]. Robust solution approaches for the COSP with scenario-dependent problem parameters were studied in [19,20]. A stochastic version of the COSP where orders arrive dynamically and their inter-arrival times form a Poisson process has been investigated by Zhao et al. in [21]. Furthermore, Lin et al. [22] consider the COSP with one machine, sequence independent setup times and the objective of minimizing a linear combination of total tardiness and total holding costs, i.e., $1 \rightarrow 0|ST_{nsd}|\alpha\sum HC_i+(1-\alpha)\sum T_i$. More recently, the COSP with a serial-batch machine was investigated in [23]. Furthermore, Hsu and Liu [24] studied the COSP for different performance indicators in order to reduce the stock level of finished goods in the job shop environment. The interested reader is referred to the review paper from Framinan et al. for an overview of configurations of the COSP [3].

Even though the problem $Fm \rightarrow 0|prmu|\sum C_i$ has not been studied before, related problems can be found in the literature. The minimization of the makespan and the total completion time of the COSP in a flow shop environment with exactly two machines was studied by Yang [25]. The author derived problem properties and developed a modification of Johnson's Algorithm to solve the problem. However, the study is limited to the two machine case. Chen et al. [26] studied manufacturing synchronization in a hybrid flow shop environment with dynamically arriving orders. The objective is to minimize the sum of the longest waiting durations of the orders. As the orders arrive dynamically and are unknown until they arrive, their solution algorithm consists of a periodic scheduling policy together with a modified genetic algorithm.

The distributed permutation flow shop problem in accordance with the COSP for minimizing the makespan over all factories, i.e., $DF \rightarrow 0|prmu|C_{max}$, is addressed by Meng et al. [27]. Here, all jobs of an order have to be processed in the same factory, but it is not required that all jobs of an order are processed consecutively. Their developed heuristics are considering both, assigning the orders to the factories and scheduling the jobs inside the factories. Even though the problem configuration seems similar to our considered problem, the implication of the customer order constraint in Meng et al. is different. Jobs belonging to the same order must be processed in the same factory. However, since the objective in Meng et al. is to minimize the makespan, i.e., the maximum completion time of all orders and hence, also of all jobs $\left(C_{max} = max_{i\in I}\{C_i\} = max_{i\in I, j\in J_i}\{C_{ij}\}\right)$, the sequencing problem inside a single factory becomes independent of the customer order constraint if the order is fixedly assigned to a factory. Therefore, the customer order constraint is more relevant for the assignment of jobs to a facility than for the actual sequencing task. Further studies which consider a

Table 1
Selected COSP literature.

Reference	Considered Problem	Model	Exact alg.	Heuristic
Erel and Ghosh [31]	$1 \rightarrow 0\|ST_{nsd}\|\sum C_i$		x (DP)	
Lin et al. [22]	$1 \rightarrow 0\|ST_{nsd}\|\alpha\sum HC_i + (1-\alpha)\sum T_i$		x (B&B)	x (SA)
Liu et al. [23]	$1 \rightarrow 0\|batch, incompt, ST_{nsd}\|\sum(\alpha_{ij}E_{ij} + \beta_i T_i)$	x		x (GA)
Roemer [7]	$DP2 \rightarrow 0\|\|\sum C_i$			
Sung and Yoon [2]	$DP2 \rightarrow 0\|\|\sum w_i C_i$			x
Wagneur and Sriskandarajah [5]	$DPm \rightarrow 0\|\|C_{max}/L_{max}/\sum C_i/\sum T_i/\sum U_i$			x
Leung et al. [8]	$DPm \rightarrow 0\|\|\sum C_i$			x (TS)
Framinan and Perez-Gonzalez [1]	$DPm \rightarrow 0\|\|\sum C_i$			x
Hoffmann et al. [11]	$DPm \rightarrow 0\|\|\sum C_i$	x		x (IGA)
Lee [12]	$DPm \rightarrow 0\|\|\sum T_i$		x (B&B)	x
Framinan and Perez-Gonzalez [13]	$DPm \rightarrow 0\|\|\sum T_i$	x		x (Math.)
Braga-Santos et al. [14]	$DPm \rightarrow 0\|\|\sum T_i$	x		x
Hoffmann et al. [32]	$DPm \rightarrow 0\|\|\sum\left(\sum E_{ij} + mT_i\right)$	x		x (IGA)
Wu et al. [33]	$DPm \rightarrow 0\|\|\sum C_i, T_{max}$		x (B&B)	x (IGA, PSO)
Wu et al. [34]	$DPm \rightarrow 0\|r_i\|\sum w_i C_i$		x (B&B)	x (IGA)
Lin et al. [15]	$DPm \rightarrow 0\|r_i\|\sum w_i U_i$		x (B&B)	x (ABC)
Prata et al. [16]	$DPm \rightarrow 0\|ST_{sd}\|C_{max}$	x		x (Math.)
Prata et al. [17]	$DPm \rightarrow 0\|ST_{sd}\|\sum C_i$	x		x (DE)
Li et al. [19]	$DPm \rightarrow 0\|r_i, scenario\|\max_{s\in\{1,2\}}\{\sum w_i U_i^{(s)}\}$		x (B&B)	x (GA)
Yang and Posner [9]	$Pm \rightarrow 0\|\|\sum C_i$			x
Shi et al. [18]	$Pm \rightarrow 0\|batch, incompt\|C_{max}/\sum w_i C_i$	x	(x)	x (DE)
Xu et al. [10]	$Rm \rightarrow 0\|split\|\sum C_i$	x		x
Yang [25]	$F2 \rightarrow 0\|\|C_{max}/\sum C_i$		(x)	x
Çetinkaya and Yozgat [35]	$F2 \rightarrow 0\|ST_{nsd}, prmu, ls\|\sum C_i$	x		x
Meng et al. [27]	$DF \rightarrow 0\|prmu\|C_{max}$	x		x (VNS, ABC, IGA)
This paper	$Fm \rightarrow 0\|prmu\|\sum C_i$	x		x (IGA)

distributed permutation flow shop environment and include a customer order constraint can be found in [28–30].

Xiong et al. [36] studied a problem which they called the precast supply chain scheduling problem in the context of the COSP. Jobs of orders had to pass through exact 9 stages, some of which could process only one job at a time (sequential stages) and the others could process multiple jobs simultaneously (parallel stages). The last stage was transportation, which was executed at an exact integer hour. Consequently, the jobs of an order were already gathered after the second last stage. The derived properties and algorithms based largely on the findings of an analysis of real world data, which made particular reductions and algorithm decision rules reasonable. Even if they provided interesting results, the transferability to our general case is very limited due to this approach. Another closely related recent paper is the study from Çetinkaya and Yozgat [35]. They investigated the COSP in a two machine flow shop environment with lot streaming, where the same products ordered by the different customers have to be processed consecutively. After providing some proofs for problem properties and a mathematical programming model, the authors presented an effective heuristic with four phases.

We provide an overview of the different configurations of the COSP which were considered in the literature in Table 1. Note that not all mentioned papers are listed in the table as the respective problem configurations do not match well with the classical three-field scheduling notation, are not defined as COSP or because the respective paper is only a revisitation of an earlier paper. As shown in the table, the literature lacks a paper that addresses the m-machine permutation flow shop environment, even though it is a prominent setting in scheduling literature. We chose the objective of minimizing the total completion time because it is one of the classical objectives in scheduling research, which is also shown in Table 1.

Besides the considered problem configuration, we also present the investigated solution approaches of the papers. An 'x' in the model-column indicates that a mathematical program that can be solved by a solver is given in the respective paper. If the authors developed an algorithm that can solve the respective problem guaranteed optimally, we mark it in the column *Exact alg.* Furthermore, we indicate if the algorithm follows a general framework, e.g., Branch-and-Bound (B&B) or Dynamic Programming (DP). Note that the exact algorithms in Yang [25] and Shi et al. [18] run in polynomial time but solve the problems $F2 \rightarrow 0||C_{max}$ and $Pm \rightarrow 0|batch, incompt|C_{max}$ respectively. In the last column, *Heuristics*, we indicate if heuristic solution approaches were developed. We also show if one or all of the developed heuristics follow a predefined framework, such as Simulated Annealing (SA), Genetic Algorithm (GA), Tabu Search (TS), IGA, Particle Swarm Optimization (PSO), Artificial Bee Colony (ABC), Differential Evolution (DE) or Variable Neighborhood Search (VNS), or are a Matheuristic (Math.).

As most scheduling problems are NP-hard, much scheduling research focuses on generating effective heuristics for solving the addressed problem, which can also be seen by the presented literature. A heuristic that attained much attention in recent years is the IGA. In a systematic literature review, Zhao et al. [4] present 137 papers which study the IGA for flow shop scheduling which shows their good performance for this machine environment. Another reason for choosing the IGA in our study is its adaptability. In the basic framework, the IGA consists of five (or four) steps, i.e., initialization, local search which is non-compulsory, destruction, construction and acceptance. Each step is a function that can be designed in various ways. The last four mentioned steps are repeated until a termination condition is met [4].

In addition to flow shop scheduling problems, the IGA has also been applied in the context of the COSP. Wu et al. [33] used an IGA to solve $DPm \rightarrow 0||\sum C_i, T_{max}$, where it outperformed other algorithms, e.g., a particle swarm optimization algorithm, while it was applied in [34] for the problem $DPm \rightarrow 0|r_i|\sum w_i C_i$. Furthermore, Hoffmann et al. [11] developed an IGA, which showed good performance for the basic configuration of the COSP, i.e., $DPm \rightarrow 0||\sum C_i$. More recently, multiple IGAs which have an additional refinement step were developed in [32] for the problem $DPm \rightarrow 0||\sum\left(\sum E_{ij} + mT_i\right)$.

The schedules which are modified by the IGAs in the first three mentioned papers of the last paragraph were represented by a sequence of orders. Consequently, the position of a job on a machine corresponded to the position of the order in the sequence. This is reasonable as by the mentioned proposition from Leung et al. [8], there exists an optimal

Table 2
Notation.

Parameters	
n	number of customer orders
m	number of machines
o	number of jobs per order
p_{ijk}	processing time of job j of order i on k
TP_{ij}	total processing time of job j of order i
Decision variables	
x_{ijh}	binary decision variable that becomes 1 if job j of order i is at position h on all machines, and 0 otherwise
C_i	completion time of order i
C_{ijk}	completion time of job j of order i on k
Notation for the algorithms	
d, d_1, d_2	destruction sizes
y, z, d_{min}, d_{max}, d_{max1}, d_{max2}	algorithm parameters
π, π_d, $\pi_{n\text{-}o\text{-}d}$	sequence of jobs
π_{orders}, π'_{orders}, $\pi_{orders:d}$, $\pi_{orders:n-d}$	sequence of orders
$\pi_{jobs,i}$, $\pi'_{jobs,i}$, $\pi^*_{jobs,i}$, $\pi_{jobs:d,i}$, $\pi_{jobs:o-d,i}$	job sequence of order i
π_{jobs}, π'_{jobs}, π^*_{jobs}, $\pi_{jobs:d}$, $\pi_{jobs:o-d}$	set of job sequences of the orders
Π, Π', Π'', Π^*	feasible schedule
$F(\cdot)$	objective function value
$C^{max}(\cdot)$	makespan of the job sequence

schedule for various problem configurations in the dedicated machine environment, where each order is processed in the same sequence on each machine, and the length of the sequence is the number of orders. However, this schedule representation is not applicable for the flow shop environment. At the same time, a schedule representation as a sequence of all jobs of all orders is computationally intensive when used in an IGA due to the length of the sequence. We will address this issue in the following sections.

3. Problem description

3.1. Problem definition

In the following, we define the considered problem formally. There are n orders where each order consists of exactly o jobs. Each of the jobs has to be processed on all machines and the machine sequence is the same for each job. We denote the number of machines with m and the numeration of the machines is equivalent to the machine sequence of production. As we consider a permutation flow shop, all jobs have to be processed in the same sequence on all machines which means a job cannot pass another job between the machines and preemptions are not allowed. Furthermore, simultaneous production of one job on multiple machines is prohibited. Job j of order i on machine k has a processing time p_{ijk} and, based on the schedule, a completion time C_{ijk}. By setting the respective processing times to zero, it is possible to generalize the problem to the case where not all orders have the same number of jobs, and where not all jobs have to be processed on each machine. Note that in the latter case, the jobs still cannot pass each other in the shop since we still consider a permutation flow shop. Release dates and setup times are not considered. The completion time of order i is defined by the completion time of the last scheduled job of this order on the last machine, i.e., $C_i = \max_{1\leq j\leq o}\{C_{ijm}\}$. The objective is to minimize the sum of completion times of all orders. An overview of the used notation in this paper can be found in Table 2.

For the purpose of clarity, a feasible schedule for an exemplary problem instance is given next. In this instance, there are 2 machines, 2 orders and 3 jobs per order, i.e., $m = 2$, $n = 2$ and $o = 3$. The processing times of this instance are stated in Table 3. Here, O_i represents order i, J_j job j of the corresponding order, and M_k machine k.

A feasible solution of this problem is the schedule with the job sequence $\{(1,1),(2,1),(2,2),(2,3),(1,2),(1,3)\}$ where for a job (i,j), i is the order number and j the job of the order. As can be seen by (1,1) and the following jobs of order 2, the jobs of a single order do not have to be processed one after another. The Gantt chart of this schedule can be found in Fig. 1. The job and order numbers of the jobs are labeled as well as some of the completion times.

Table 3
Processing times of the exemplary problem instance.

p_{ijk}	J_1		J_2		J_3	
	M_1	M_2	M_1	M_2	M_1	M_2
O_1	2	3	5	3	6	2
O_2	3	4	3	3	4	2

By this example we also introduce the terms *forced* and *unforced idle times* as well as *non-delay schedule*. As can be seen in Fig. 1, there are idle times between some jobs, e.g., between $(2,3)$ and $(1,2)$ on machine 2. However, these idle times are forced by the flow shop property according to which a job can only be processed on the next machine if it is finished on the current machine. In contrast, unforced idle times are actively inserted by the decision maker and occur if a machine is kept idle despite a job is waiting before the machine for processing. If a schedule has no unforced idle times, the schedule is called non-delay (see [37] for further explanation). An example for an unforced idle time for the problem instance and job sequence above would be the following. Job $(2,3)$ could start at time 13 on machine 2, despite being finished at time 12 on machine 1 and the previously scheduled job $(2,2)$ finished at time 12 on machine 2. The remaining operations are as in Fig. 1. Consequently, machine 2 would be kept idle for one time unit even though job $(2,3)$ would wait before the machine for processing.

3.2. Problem properties

We are presenting some important problem properties in the following.

Proposition 1. *The problem $Fm \to 0|prmu|\sum C_i$ is NP-hard, even for the two machine case.*

Proof. The proof follows the ideas of a proof presented in [25]. Garey et al. [38] proved that the problem $F2||\sum C_i$ is NP-hard. A well known property of $F2||\sum C_i$ is that there exists always an optimal solution where the jobs are processed in the same sequence on each machine [39]. As a result, $F2|prmu|\sum C_i$ and hence, $Fm|prmu|\sum C_i$, are also NP-hard. For the case $o = 1$, i.e., each order includes just one job, the problem $F2 \to 0|prmu|\sum C_i$ is equal to $F2|prmu|\sum C_i$. Therefore, $F2 \to 0|prmu|\sum C_i$ and hence, $Fm \to 0|prmu|\sum C_i$, are also NP-hard.

Lemma 2. *There exists an optimal schedule for each instance of $Fm \to 0|prmu|\sum C_i$ without unforced idle times, i.e., one of the non-delay schedules is optimal.*

Proof. Inserting an unforced idle time in front of a job cannot reduce the completion time of this job and, due to the permutation schedule requirement, not of any other following job. The completion times of the previously scheduled jobs are not affected by the idle time. Since $C_i = \max_{1\leq j\leq o}\{C_{ijm}\}$, the completion time of any order cannot be reduced either and therefore, no smaller total completion time of the orders can be found.

Remark 1. Note that there are problem instances with multiple optimal solutions, and some of these optimal solutions may contain unforced idle times.

Lemma 3. *There are instances of the problem $Fm \to 0|prmu|\sum C_i$ where it is not possible to schedule all jobs of an order consecutively to obtain an optimal solution. This even applies to the problem size $n = 2$, $o = 2$, $m = 2$.*

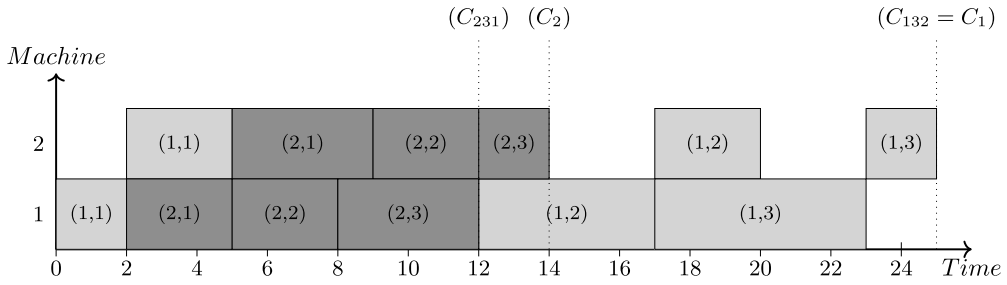


Fig. 1. Gantt chart of the example.

Table 4
Processing times of the counter example.

p_{ijk}	J_1		J_2	
	M_1	M_2	M_1	M_2
O_1	10	11	7	2
O_2	11	17	1	17

Table 5
Total completion times of the sequences where the jobs of an order are scheduled consecutively.

Job sequence	Total completion time
$\{(1,1),(1,2),(2,1),(2,2)\}$	85
$\{(1,1),(1,2),(2,2),(2,1)\}$	80
$\{(1,2),(1,1),(2,1),(2,2)\}$	90
$\{(1,2),(1,1),(2,2),(2,1)\}$	90
$\{(2,1),(2,2),(1,1),(1,2)\}$	103
$\{(2,1),(2,2),(1,2),(1,1)\}$	103
$\{(2,2),(2,1),(1,1),(1,2)\}$	83
$\{(2,2),(2,1),(1,2),(1,1)\}$	83

Proof. This property can be shown by the counter example with the processing times given in Table 4.

The job sequence with the lowest total completion time is $\{(2,2),(1,2),(1,1),(2,1)\}$ where i represents the order and j the job of the order in (i,j). The corresponding total completion time is 79. The total completion times of the sequences where all jobs of an order are scheduled consecutively are given in Table 5.

This counter example proves the Lemma since there exists an instance where there is no optimal schedule where the jobs of each order are scheduled consecutively.

Remark 2. A further reduction of the problem size $n = 2$, $o = 2$, $m = 2$ is not in the scope of this research. As mentioned, $o = 1$ would result in the problem $Fm|prmu|\sum C_i$ which is well studied. If $n = 1$, the problem becomes the well known problem $Fm|prmu|C_{max}$. In the case $m = 1$, we get the problem $1 \to 0|prmu|\sum C_i$, which can be solved by an extension of the SPT-rule to the customer order case [31].

Lemmas 2 and 3 give important implications for the representation of a complete schedule Π in this study: a solution or complete schedule Π is represented by a sequence of all jobs of all orders and unforced idle times are not considered. An example for this was given earlier in Section 3.1. The solution representation is used in the proposed algorithms and in the MINLP formulation. For the MINLP, we use the already introduced decision variables C_{ijk} and C_i. Furthermore, we use the binary decision variable x_{ijh} which indicates if job j from order i is in position h on each machine (see Table 2).

$$\text{minimize: } \sum_{i=1}^{n} C_i \tag{1}$$

subject to:

$$\sum_{h=1}^{n\cdot o} x_{ijh} = 1 \;\forall\; i = 1,\dots,n;\; j = 1,\dots,o \tag{2}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{o} x_{ijh} = 1 \;\forall\; h = 1,\dots,n\cdot o \tag{3}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{o} x_{ij(h-1)}\cdot C_{ijk} \le \sum_{i=1}^{n}\sum_{j=1}^{o} x_{ijh}\cdot\left(C_{ijk} - p_{ijk}\right) \;\forall \quad h = 2,\dots,n\cdot o;\; k = 1,\dots,m \tag{4}$$

$$C_{ij(k-1)} \le C_{ijk} - p_{ijk} \;\forall\; i = 1,\dots,n;\; j = 1,\dots,o;\; k = 2,\dots,m \tag{5}$$

$$0 \le \sum_{i=1}^{n}\sum_{j=1}^{o} x_{ij1}\cdot\left(C_{ij1} - p_{ij1}\right) \tag{6}$$

$$C_{ijm} \le C_i \;\forall\; i = 1,\dots,n;\; j = 1,\dots,o \tag{7}$$

$$x_{ijh} \in \{0;1\} \;\forall\; i = 1,\dots,n;\; j = 1,\dots,o;\; h = 1,\dots,n\cdot o \tag{8}$$

The objective of minimizing the total completion time is defined in Eq. (1). By Eqs. (2) and (3) it is guaranteed that each job of each order is assigned to exactly one sequence position and vice versa. Eq. (4) defines for each machine that a job in position $h-1$ has to be finished before the processing of the following job (in position h) can start. Furthermore, it is guaranteed by Eq. (5) that a job cannot be processed on machine k until processing has finished on the previous machine $k-1$. As there is no x_{ij0} and C_{ij0} defined, Eq. (6) gives the starting condition for the first machine and position. The definition of the completion time of an order C_i can be found in Eq. (7) and the definition of the binarity of x_{ijh} in Eq. (8).

4. Algorithm description

As the problem is NP-hard and the number of jobs increases with both, increasing n and o, we developed four heuristics for this problem. The heuristics are based on a variant of the IGA which is presented in [11] and is called IGN in the respective paper.

Initial solution

Each of the heuristics generates an initial solution with a function we call *MultipleNEH*. The function has two parts and the pseudocode of the function can be found in Algorithm 1. First, the NEH heuristic [40] is applied to the jobs of each order separately. For the example in Section 3.1, this means that the jobs of the first order are sorted first and then the jobs of the other order, each time without considering the jobs of the other order in the shop. The NEH heuristic is applied in the following way. For each order, the jobs are sorted in ascending sequence of their total production time $TP_{ij} = \sum_{k=1}^{m} p_{ijk}$. The first two jobs from the sorted list are selected and brought into the sequence that generates the lowest makespan when only these two jobs are in the shop. Subsequently, the next job of the sorted list is inserted into every possible position of the existing job sequence and is saved at the position for which the minimum makespan results. This is repeated until all jobs of the order are placed. An exemplary job sequence $\pi_{jobs,1} = \{(1,1),(1,2),(1,3)\}$ for the jobs of order 1 of the example in

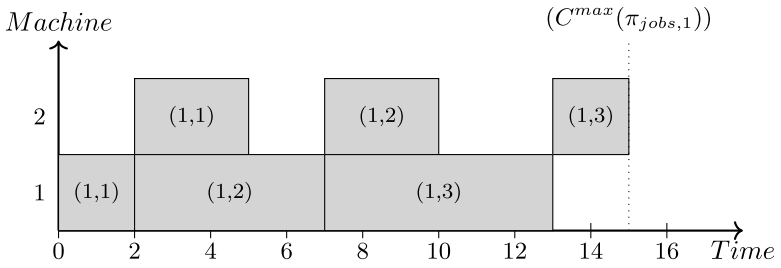


Fig. 2. Gantt chart for the job sequence of a single order.

Algorithm 1 MultipleNEH function

```
procedure MultipleNEH
    π_jobs ← {}
    for i ← 1 to n do
        for j ← 1 to o do
            TP_ij ← Σ_{k=1}^{m} p_ijk
        end for
        Sort the jobs in ascending order of TP in list
        π_jobs,i ← {}
        Insert job at position 1 of list in π_jobs,i
        for a ← 2 to o do
            for b ← len(π_jobs,i) + 1 to 1 do
                π_jobs,i,b ← insert job at position a of list in π_jobs,i at position b
            end for
            π_jobs,i ← π_jobs,i,b with minimum C^max(π_jobs,i,b)
        end for
        insert π_jobs,i in π_jobs
    end for
    for i ← 1 to n do
        C_i^max ← max_{1≤j≤o}{C_ijm(π_jobs,i)}
    end for
    Sort the orders in ascending order of C_i^max in list
    π_orders ← {}
    Insert order at position 1 of list in π_orders
    for a ← 2 to n do
        for b ← len(π_orders) + 1 to 1 do
            π_orders,b ← insert order at position a of list in π_orders at position b
            π_b ← π_orders,b, π_jobs
        end for
        π_orders ← π_orders,b with minimum F(π_b)
    end for
    Π ← π_orders, π_jobs
    return Π, π_orders, π_jobs
end procedure
```

Section 3.1 is given in Fig. 2. We also marked the makespan of $\pi_{jobs,1}$ in the figure.

After generating the job sequences for each order, we generate an order sequence π_{orders} in the second part of MultipleNEH. By this it is determined which job sequence of a single order $\pi_{jobs,i}$ is processed in which position of the complete schedule Π. First, the orders are sorted in ascending sequence of the makespan of their job sequences. In the next step, the first two orders from the resulting list are selected and their job sequences are brought into the sequence which minimizes the total completion time of the orders. Note, that the jobs of an order are still being processed consecutively, and it is only determined whether $\pi_{jobs,i}$ is before $\pi_{jobs,j}$ or vice versa. Subsequently, the next order from the list is chosen and the corresponding job sequence is placed in each position of the job sequences and is saved at the position for which the minimum total completion time resulted. This repeats until the job sequences of all orders are placed. The result of the second part of MultipleNEH is a sequence of orders π_{orders} which represents the positions of the job sequences of the orders in the resulting schedule. Together with the job sequences of the single orders $\pi_{jobs} = \{\pi_{jobs,1}, \pi_{jobs,2}, \dots, \pi_{jobs,n}\}$ we can generate a complete schedule Π. For example, with $\pi_{orders} = \{2,1\}$ and $\pi_{jobs} = \{\pi_{jobs,1}, \pi_{jobs,2}\} = \{\{(1,1),(1,2),(1,3)\},\{(2,2),(2,1),(2,3)\}\}$ we get the schedule $\Pi = \{(2,2),(2,1),(2,3),(1,1),(1,2),(1,3)\}$. The schedule can be found in Fig. 3. The total completion time of this schedule is $F(\Pi) = \sum_{i=1}^{n} C_i(\Pi) = 25 + 12 = 37$.

IGA-string algorithm

Our first heuristic is a straight forward approach by applying the IGA to the complete job sequence Π. We call this heuristic IGA-String as the functions of the IGA modify solely the sequence of all jobs of all orders, i.e. one string of jobs. We refer to Algorithm 2 for the pseudocode of this heuristic.

Algorithm 2 General procedure of IGA-String

```
procedure IGA-String(d_min, d_max, y, z)
    Π ← MultipleNEH()
    d ← d_min
    Π ← MultipleInsertion(Π, z, d, d_min)
    Π* ← Π
    while time limit not exceeded do
        π_d, π_{n·o−d} ← Destruction(Π, d)
        Π' ← Construction(π_d, π_{n·o−d})
        Π' ← MultipleInsertion(Π', z, d, d_min)
        Π*, Π, d ← Acceptance(Π*, Π, Π', d, d_min, d_max, y)
    end while
    return Π*, F(Π*)
end procedure
```

After generating a first feasible schedule with MultipleNEH, the destruction size d is set to d_{min} in IGA-String. During *MultipleInsertion*, $\left\lfloor z \cdot \frac{d}{d_{min}} \right\rfloor$ times a single job is taken out of the job sequence and placed in each position of the job sequence. Each time, the job is saved at the position for which the minimum total completion time resulted. Note that it is not possible to choose the same job multiple times per execution of MultipleInsertion. The pseudocode of this function is given in Algorithm 3. Subsequently, the obtained schedule is saved as the best found schedule Π^*.

The following procedure is repeated until a given time limit is exceeded. In *Destruction*, d random jobs are taken out of the job sequence and are stored in π_d. The remaining jobs are stored in the partial schedule $\pi_{n\cdot o-d}$ in the same sequence as they appear in Π. Subsequently, a new schedule Π' is generated by *Construction*. First, the jobs in π_d are sorted in ascending order by their total production time $TP_{ij} = \sum_{k=1}^{m} p_{ijk}$. Then, the jobs in the sorted string π_d are reinserted one after another at the position in $\pi_{n\cdot o-d}$ which minimizes the total order completion time of the job sequence. The pseudocodes of the Destruction and Construction function can be found in Algorithm 4 and Algorithm 5 respectively.

After applying the local search function MultipleInsertion to the schedule Π', the resulting schedule is evaluated by the function *Acceptance* (see Algorithm 6 for the pseudocode). This function determines the schedule Π as well as the destruction size d for the next iteration, and checks if a new best solution was found. The schedule Π' is

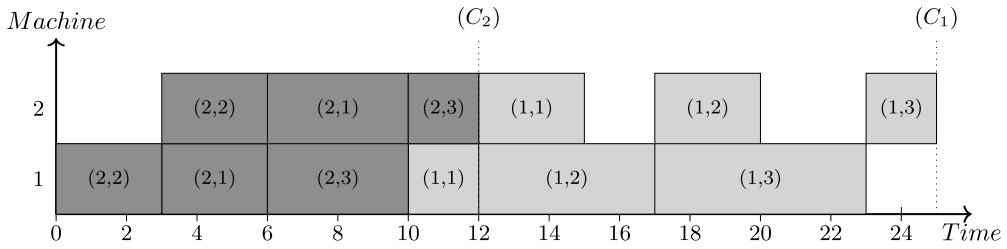


Fig. 3. Gantt chart of the schedule $\Pi = \{(2,2),(2,1),(2,3),(1,1),(1,2),(1,3)\}$.

Algorithm 3 Multiple insertion in IGA-String

procedure MULTIPLEINSERTION(Π, z, d, d_{min})
  $\Pi_{copy} \leftarrow \Pi$
  for $a \leftarrow 1$ **to** $\left\lfloor z \cdot \frac{d}{d_{min}} \right\rfloor$ **do**
    take a random job out of Π_{copy} and insert it in *list*
  end for
  for $a \leftarrow 1$ **to** $\left\lfloor z \cdot \frac{d}{d_{min}} \right\rfloor$ **do**
    $p \leftarrow$ job at position a of *list*
    $\pi \leftarrow \Pi \backslash p$
    for $b \leftarrow n \cdot o$ **to** 1 **do**
      $\pi_b \leftarrow$ insert p in π at position b
    end for
    $\Pi \leftarrow \pi_b$ with minimum $F(\pi_b)$
  end for
  return Π
end procedure

Algorithm 4 Destruction in IGA-String

procedure DESTRUCTION(Π, d)
  $\pi_{n\cdot o-d}, \pi_d \leftarrow \Pi, \{\}$
  for $a \leftarrow 1$ **to** d **do**
    take a random job out of $\pi_{n\cdot o-d}$ and append it to π_d
  end for
  return π_d, $\pi_{n\cdot o-d}$
end procedure

Algorithm 5 Construction in IGA-String

procedure CONSTRUCTION(π_d, $\pi_{n\cdot o-d}$)
  for $a \leftarrow 1$ **to** $len(\pi_d)$ **do**
    $(i,j) \leftarrow \pi_d(a)$ ▷ job at position a in π_d
    $TP_{ij} \leftarrow \sum_{k=1}^{m} p_{ijk}$
  end for
  Sort the jobs in ascending order of TP in π_d
  while $0 < len(\pi_d)$ **do**
    $p \leftarrow \pi_d(1)$
    for $a \leftarrow len(\pi_{n\cdot o-d}) + 1$ **to** 1 **do**
      $\pi_a \leftarrow$ insert p in $\pi_{n\cdot o-d}$ at position a
    end for
    remove p from π_d
    $\pi_{n\cdot o-d} \leftarrow \pi_a$ with minimum $F(\pi_a)$
  end while
  $\Pi' \leftarrow \pi_{n\cdot o-d}$
  return Π'
end procedure

accepted as the new schedule for the next iteration if $F(\Pi') \leq F(\Pi)$ and as the new best found schedule if $F(\Pi') < F(\Pi^*)$. Furthermore, if $F(\Pi') < F(\Pi)$, d is set to d_{min}, otherwise d is increased by 1 if the upper limit d_{max} is not reached, i.e., $d < d_{max}$. A constructed schedule Π' can also be taken as the schedule for the next iteration for the case $F(\Pi') > F(\Pi)$, but only if $q \leq exp(-y \cdot \frac{d_{min}}{d} \cdot \frac{F(\Pi')-F(\Pi)}{F(\Pi)})$ holds, where q is a random number between 0 and 1, drawn each iteration.

Algorithm 6 Acceptance in IGA-String

procedure ACCEPTANCE($\Pi^*, \Pi, \Pi', d, d_{min}, d_{max}, y$)
  if $F(\Pi') < F(\Pi)$ **then**
    $d, \Pi \leftarrow d_{min}, \Pi'$
    if $F(\Pi') < F(\Pi^*)$ **then**
      $\Pi^* \leftarrow \Pi'$
    end if
  else if $F(\Pi') = F(\Pi)$ **then**
    $\Pi \leftarrow \Pi'$
    if $d < d_{max}$ **then**
      $d \leftarrow d + 1$
    end if
  else
    if $q \leq exp(-y \cdot \frac{d_{min}}{d} \cdot \frac{F(\Pi')-F(\Pi)}{F(\Pi)})$ **then**
      $\Pi \leftarrow \Pi'$
    end if
    if $d < d_{max}$ **then**
      $d \leftarrow d + 1$
    end if
  end if
  return Π^*, Π, d
end procedure

Table 6
Schedule representation in tabular form.

Order number	Job sequence		
2	(2,2)	(2,1)	(2,3)
1	(1,1)	(1,2)	(1,3)

IGA-matrix algorithms in general

The downside of IGA-String is the high computational effort for each iteration as the whole job sequence Π is modified. This could result in an inappropriate exploration of the solution space in a given time limit. Consequently, with most of their functions, the other three heuristics are not modifying the complete sequence of all jobs of all orders but only the job sequences of each single order $\pi_{jobs,i}$ and the sequence of orders π_{order} individually. This is done by applying two destruction, construction and local search functions per iteration, one combination for modifying $\pi_{jobs,i}$ and one for modifying π_{orders}. The resulting schedule after these functions is a job sequence where the jobs of each order are scheduled consecutively. Note, that this also applies for the resulting schedule of the function MultipleNEH. Since this schedule can be written in tabular form, where in a row, the job sequence of a single order is defined, and the row number indicates the position of the job sequence of an order in the complete schedule, the three heuristics are named IGA-Matrix. An example for this way of representing the schedule is given in Table 6 for the example schedule from the beginning of this section, i.e., $\Pi = \{(2,2),(2,1),(2,3),(1,1),(1,2),(1,3)\}$. The idea of considering the job sequences of the orders separately is similar to the concept of group scheduling [41,42].

The main differences between the three IGA-Matrix algorithms are the following. IGA-Matrix1 and IGA-Matrix2 are using two acceptance functions. The first one evaluates the job sequences of the single orders

$\pi_{jobs,i}$, the second one the order sequence π_{orders}. This is based on the idea that we first want to find appropriate job sequences of the single orders, independent of the jobs of the other orders, and then adjust the sequence of these job sequences accordingly. Furthermore, due to Lemma 3, there are optimal schedules where the jobs of an order are not scheduled consecutively. Consequently, IGA-Matrix2 and IGA-Matrix3 are using an additional local search function for further refinement, where jobs of an order are scheduled actively between jobs of another order.

IGA-Matrix1 algorithm

Algorithm 7 General procedure of IGA-Matrix1

procedure IGA-MATRIX1($d_{min}, d_{max1}, d_{max2}, y, z$)
  $\Pi, \pi_{orders}, \pi_{jobs} \leftarrow MultipleNEH()$
  $d_1, d_2, \Pi^* \leftarrow d_{max1}, d_{min}, \Pi$
  $\pi_{jobs} \leftarrow MultipleInsertionJobs(\pi_{jobs}, z, d_1, d_{min})$
  $\pi^*_{jobs} \leftarrow \pi_{jobs}$
  $\Pi, \pi_{orders} \leftarrow MultipleInsertionOrder(\pi_{orders}, \pi^*_{jobs}, z, d_2, d_{min})$
  if $F(\Pi) < F(\Pi^*)$ **then**
    $\Pi^* \leftarrow \Pi$
  end if
  while time limit not exceeded **do**
    $\pi_{jobs:d}, \pi_{jobs:o-d} \leftarrow DestructionJobs(\pi_{jobs}, d_1)$
    $\pi'_{jobs} \leftarrow ConstructionJobs(\pi_{jobs:d}, \pi_{jobs:o-d})$
    $\pi'_{jobs} \leftarrow MultipleInsertionJobs(\pi'_{jobs}, z, d_1, d_{min})$
    $\pi^*_{jobs}, \pi_{jobs}, d_1 \leftarrow AccJobs(\pi^*_{jobs}, \pi'_{jobs}, \pi_{jobs}, d_1, d_{min}, y)$
    $\Pi \leftarrow \pi_{orders}, \pi^*_{jobs}$
    if $F(\Pi) < F(\Pi^*)$ **then**
      $\Pi^* \leftarrow \Pi$
    end if
    $\pi_{orders:d}, \pi_{orders:n-d} \leftarrow DestructionOrder(\pi_{orders}, d_2)$
    $\pi'_{orders} \leftarrow ConstructionOrder(\pi_{orders:d}, \pi_{orders:n-d}, \pi^*_{jobs})$
    $\Pi', \pi'_{orders} \leftarrow MultipleInsertionOrder(\pi'_{orders}, \pi^*_{jobs}, z, d_2, d_{min})$
    $\Pi^*, \pi_{orders}, d_2 \leftarrow AccOrder1(\Pi, \Pi^*, \Pi', \pi'_{orders}, \pi_{orders}, d_2, d_{min}, d_{max2}, y)$
  end while
  return $\Pi^*, F(\Pi^*)$
end procedure

Next, we are describing the three IGA-Matrix functions in detail. The pseudocode of IGA-Matrix1 can be found in Algorithm 7. After generating a first schedule as well as job sequences of the single orders and an order sequence with MulitpleNEH, the destruction size for the job sequences d_1 is set to d_{max1}, and the destruction size for the order sequence d_2 to d_{min}. The destruction size d_1 decreases over the iterations, while d_2 behaves similarly to d in IGA-String. As a result, the modification of the job sequences of single orders is prioritized in the first iterations, while the order sequence can be adjusted to comparatively constant job sequences in later iterations. Furthermore, the generated schedule is marked as the best found schedule.

Two local search functions, one for the job sequences and one for the order sequence, are executed afterwards. In *MultipleInsertionJobs*, $\left\lfloor z \cdot \frac{d_1}{d_{min}} \right\rfloor$ times a job is taken out of the job sequence $\pi_{jobs,i}$ and is reinserted into every possible position of the sequence. Each time it is saved at the position for which the minimum makespan results. This is repeated for the job sequence of each order. The resulting set of job sequences π_{jobs} is additionally marked as π^*_{jobs} afterwards. This set is also used in the following local search function *MultipleInsertionOrder*. Here, $\left\lfloor z \cdot \frac{d_2}{d_{min}} \right\rfloor$ times an order is removed from the order sequence π_{orders} and subsequently reinserted into every possible position of the order sequence. The order sequence for which the minimum total order completion time of the corresponding schedule, i.e., the schedule which results from π_{jobs} and the respective π_{orders}, is saved and used for the next iteration each time. The pseudocodes of the two local search functions can be found in Algorithms 8 and 9. The resulting schedule from MultipleInsertionOrder Π is compared with the schedule Π^*, and if $F(\Pi) < F(\Pi^*)$, the schedule Π is saved as best found schedule Π^*.

Algorithm 8 Multiple insertion of jobs in IGA-Matrix algorithms

procedure MULTIPLEINSERTIONJOBS(π_{jobs}, z, d_1, d_{min})
  for $i \leftarrow 1$ **to** n **do**
    $\pi_{copy} \leftarrow \pi_{jobs,i}$
    for $a \leftarrow 1$ **to** $\left\lfloor z \cdot \frac{d_1}{d_{min}} \right\rfloor$ **do**
      Take a random job out of π_{copy} and insert it in *list*
    end for
    for $a \leftarrow 1$ **to** $\left\lfloor z \cdot \frac{d_1}{d_{min}} \right\rfloor$ **do**
      $p \leftarrow$ job at position a of *list*
      $\pi_{jobs,i,O} \leftarrow \pi_{jobs,i} \backslash p$
      for $b \leftarrow o$ **to** 1 **do**
        $\pi_{jobs,i,b} \leftarrow$ insert p in $\pi_{jobs,i,O}$ at position b
      end for
      $\pi_{jobs,i} \leftarrow \pi_{jobs,i,b}$ with minimum $C^{max}(\pi_{jobs,i,b})$
    end for
  end for
  return π_{jobs}
end procedure

Algorithm 9 Multiple insertion of orders in IGA-Matrix algorithms

procedure MULTIPLEINSERTIONORDER(π_{orders}, π_{jobs}, z, d_2, d_{min})
  $\pi_{copy} \leftarrow \pi_{orders}$
  for $a \leftarrow 1$ **to** $\left\lfloor z \cdot \frac{d_2}{d_{min}} \right\rfloor$ **do**
    take a random order out of π_{copy} and insert it in *list*
  end for
  for $a \leftarrow 1$ **to** $\left\lfloor z \cdot \frac{d_2}{d_{min}} \right\rfloor$ **do**
    $p \leftarrow$ order at position a of *list*
    $\pi_{orders,O} \leftarrow \pi_{orders} \backslash p$
    for $b \leftarrow n$ **to** 1 **do**
      $\pi_{orders,b} \leftarrow$ insert p in $\pi_{orders,O}$ at position b
      $\Pi_b \leftarrow \pi_{orders,b}, \pi_{jobs}$
    end for
    $\pi_{orders} \leftarrow \pi_{orders,b}$ with minimum $F(\Pi_b)$
  end for
  $\Pi \leftarrow \pi_{orders}, \pi_{jobs}$
  return Π, π_{orders}
end procedure

Note that the comparison of Π and Π^* is made as the resulting schedule from the two local search functions might be worse than the previously obtained schedule.

Algorithm 10 Destruction of job sequences in IGA-Matrix algorithms

procedure DESTRUCTIONJOBS(π_{jobs}, d_1)
  $\pi_{jobs:o-d}, \pi_{jobs:d} \leftarrow \{\}, \{\}$
  for $a \leftarrow 1$ **to** n **do**
    $\pi_{jobs:o-d,a}, \pi_{jobs:d,a} \leftarrow \pi_{jobs,a}, \{\}$
    for $b \leftarrow 1$ **to** d_1 **do**
      take a random job out of $\pi_{jobs:o-d,a}$ and append it to $\pi_{jobs:d,a}$
    end for
    Append $\pi_{jobs:o-d,a}$ to $\pi_{jobs:o-d}$ and $\pi_{jobs:d,a}$ to $\pi_{jobs:d}$
  end for
  return $\pi_{jobs:d}, \pi_{jobs:o-d}$
end procedure

The subsequent procedure repeats until a given time limit is exceeded. In each iteration, the procedure starts with *DestructionJobs* (see Algorithm 10). For each order, d_1 random jobs are taken out of the corresponding job sequence $\pi_{jobs,i}$ and are stored in $\pi_{jobs:d,i}$. The remaining jobs from the original job sequence are stored in $\pi_{jobs:o-d,i}$ in the same sequence as they appear in $\pi_{jobs,i}$. The resulting sets of job sequences are $\pi_{jobs:d}$ and $\pi_{jobs:o-d}$ respectively. In *ConstructionJobs* (see Algorithm 11), the jobs of each job sequence $\pi_{jobs:d,i}$ are reinserted into the corresponding $\pi_{jobs:o-d,i}$. For each order, the jobs in the corresponding $\pi_{jobs:d,i}$ are sorted in ascending sequence of their total production

Algorithm 11 Construction of job sequences in IGA-Matrix algorithms

procedure ConstructionJobs($\pi_{jobs:d}$, $\pi_{jobs:o-d}$)
  $\pi'_{jobs} \leftarrow \{\}$
  for $i \leftarrow 1$ **to** n **do**
    for $a \leftarrow 1$ **to** $len(\pi_{jobs:d,i})$ **do**
      $(i, j) \leftarrow \pi_{jobs:d,i}(a)$
      $TP_{ij} \leftarrow \sum_{k=1}^{m} p_{ijk}$
    end for
    Sort the jobs in ascending order of TP in $\pi_{jobs:d,i}$
    while $0 < len(\pi_{jobs:d,i})$ **do**
      $p \leftarrow \pi_{jobs:d,i}(1)$
      for $a \leftarrow len(\pi_{jobs:o-d,i}) + 1$ **to** 1 **do**
        $\pi_{jobs:o-d,i,a} \leftarrow$ insert p in $\pi_{jobs:o-d,i}$ at position a
      end for
      remove p from $\pi_{jobs:d,i}$
      $\pi_{jobs:o-d,i} \leftarrow \pi_{jobs:o-d,i,a}$ with minimum $C^{max}(\pi_{jobs:o-d,i,a})$
    end while
    $\pi'_{jobs,i} \leftarrow \pi_{jobs:o-d,i}$, and append $\pi'_{jobs,i}$ to π'_{jobs}
  end for
  return π'_{jobs}
end procedure

time TP_{ij}. Subsequently, the first job from $\pi_{jobs:d,i}$ is inserted into every possible position of $\pi_{jobs:o-d,i}$ and saved at the position for which the minimum makespan of $\pi_{jobs:o-d,i}$ resulted. This is repeated for each job of the sorted $\pi_{jobs:d,i}$. The resulting job sequence $\pi'_{jobs,i}$ of each order is stored in π'_{jobs}.

Algorithm 12 Acceptance of job sequences in IGA-Matrix algorithms

procedure AccJobs($\pi^*_{jobs}, \pi'_{jobs}, \pi_{jobs}, d_1, d_{min}, y$)
  for $i \leftarrow 1$ **to** n **do**
    if $C^{max}(\pi'_{jobs,i}) < C^{max}(\pi_{jobs,i})$ **then**
      $\pi_{jobs,i} \leftarrow \pi'_{jobs,i}$
      if $C^{max}(\pi'_{jobs,i}) \leq C^{max}(\pi^*_{jobs,i})$ **then**
        $\pi^*_{jobs,i} \leftarrow \pi'_{jobs,i}$
      end if
    else if $C^{max}(\pi'_{jobs,i}) = C^{max}(\pi_{jobs,i})$ **then**
      $\pi_{jobs,i} \leftarrow \pi'_{jobs,i}$
      if $C^{max}(\pi'_{jobs,i}) = C^{max}(\pi^*_{jobs,i})$ **then**
        $\pi^*_{jobs,i} \leftarrow \pi'_{jobs,i}$
      end if
    else
      if $q \leq exp(-y \cdot \frac{C^{max}(\pi'_{jobs,i}) - C^{max}(\pi_{jobs,i})}{C^{max}(\pi_{jobs,i})})$ **then**
        $\pi_{jobs,i} \leftarrow \pi'_{jobs,i}$
      end if
    end if
  end for
  if $d_1 > d_{min}$ **then**
    $d_1 \leftarrow d_1 - 1$
  end if
  return $\pi^*_{jobs}, \pi_{jobs}, d_1$
end procedure

After applying MultipleInsertionJobs to π'_{jobs}, each job sequence $\pi'_{jobs,i}$ is compared with the corresponding job sequences $\pi_{jobs,i}$ and $\pi^*_{jobs,i}$ in *AccJobs* (see Algorithm 12). If the makespan of the sequence $\pi'_{jobs,i}$ is smaller than the makespan of $\pi_{jobs,i}$, the sequence $\pi'_{jobs,i}$ becomes the new $\pi_{jobs,i}$ and is used in the next iteration. Furthermore, if the makespan of $\pi'_{jobs,i}$ is also smaller than or equal to the makespan of $\pi^*_{jobs,i}$, it also becomes the new $\pi^*_{jobs,i}$. For the case that the makespan of $\pi'_{jobs,i}$ is equal to the makespan of $\pi_{jobs,i}$, $\pi'_{jobs,i}$ replaces $\pi_{jobs,i}$, and if it is also equal to the makespan of $\pi^*_{jobs,i}$, $\pi'_{jobs,i}$ replaces also this sequence. If the makespan of $\pi'_{jobs,i}$ is larger than the makespan of $\pi_{jobs,i}$, the sequence $\pi_{jobs,i}$ is replaced by $\pi'_{jobs,i}$ if the following inequation is true: $q \leq exp(-y \cdot \frac{C^{max}(\pi'_{jobs,i}) - C^{max}(\pi_{jobs,i})}{C^{max}(\pi_{jobs,i})})$, where q is a random number between 0 and 1, drawn each time. After evaluating the job sequence of each order, d_1 is decreased by 1 if d_{min} has not already been reached. The output of the function includes the job sequence π_{jobs} which is used in the next iteration, and π^*_{jobs} which is used in the remainder of the iteration for generating schedules with a given order sequence. In a next step, the schedule Π, generated by π^*_{jobs} and π_{orders}, is compared with the best found schedule Π^*, and if the total completion time of Π is lower, the schedule Π is marked as the best found schedule.

Algorithm 13 Destruction of order sequence in IGA-Matrix algorithms

procedure DestructionOrder(π_{orders}, d_2)
  $\pi_{orders:n-d}, \pi_{orders:d} \leftarrow \pi_{orders}, \{\}$
  for $a \leftarrow 1$ **to** d_2 **do**
    take a random order out of $\pi_{orders:n-d}$ and append it to $\pi_{orders:d}$
  end for
  return $\pi_{orders:d}, \pi_{orders:n-d}$
end procedure

Algorithm 14 Construction of order sequence in IGA-Matrix algorithms

procedure ConstructionOrder($\pi_{orders:d}$, $\pi_{orders:n-d}$, π_{jobs})
  Sort the orders of $\pi_{orders:d}$ in ascending order of $C^{max}(\pi_{jobs,i})$ in $\pi_{orders:d}$
  while $0 < len(\pi_{orders:d})$ **do**
    $p \leftarrow \pi_{orders:d}(1)$
    for $a \leftarrow len(\pi_{orders:n-d}) + 1$ **to** 1 **do**
      $\pi_{orders:n-d,a} \leftarrow$ insert p in $\pi_{orders:n-d}$ at position a
      $\pi_a \leftarrow \pi_{orders:n-d,a}, \pi_{jobs}$
    end for
    remove p from $\pi_{orders:d}$
    $\pi_{orders:n-d} \leftarrow \pi_{orders:n-d,a}$ with minimum $F(\pi_a)$
  end while
  $\pi'_{orders} \leftarrow \pi_{orders:n-d}$
  return π'_{orders}
end procedure

Algorithm 15 Acceptance of order sequence in IGA-Matrix1

procedure AccOrder1($\Pi, \Pi^*, \Pi', \pi'_{orders}, \pi_{orders}, d_2, d_{min}, d_{max2}, y$)
  if $F(\Pi') < F(\Pi)$ **then**
    $\pi_{orders}, d_2 \leftarrow \pi'_{orders}, d_{min}$
    if $F(\Pi') < F(\Pi^*)$ **then**
      $\Pi^* \leftarrow \Pi'$
    end if
  else if $F(\Pi') = F(\Pi)$ **then**
    $\pi_{orders} \leftarrow \pi'_{orders}$
    if $d_2 < d_{max2}$ **then**
      $d_2 \leftarrow d_2 + 1$
    end if
  else
    if $q \leq exp(-y \cdot \frac{d_{min}}{d_2} \cdot \frac{F(\Pi') - F(\Pi)}{F(\Pi)})$ **then**
      $\pi_{orders} \leftarrow \pi'_{orders}$
    end if
    if $d_2 < d_{max2}$ **then**
      $d_2 \leftarrow d_2 + 1$
    end if
  end if
  return Π^*, π_{orders}, d_2
end procedure

In the remaining part of the iteration, the sequence of orders is modified. By *DestructionOrder*, d_2 orders are taken out of π_{orders} and are stored in $\pi_{orders:d}$, while the other orders remain in their sequence in $\pi_{orders:n-d}$. The pseudocode of this function can be seen in Algorithm 13.

During *ConstructionOrder* (see Algorithm 14), the orders in $\pi_{orders:d}$ are initially sorted in ascending sequence according to the makespan

of their job sequence. Afterwards, the first order of the sorted sequence is reinserted into $\pi_{orders:n-d}$ at each position, and is saved at the place that generates the lowest total completion time for the corresponding partial schedule π, which results from the order sequence and the corresponding job sequences. This is repeated for each order from $\pi_{orders:d}$.

After using the function MultipleInsertionOrder again, we obtain the order sequence π'_{orders} and the corresponding schedule Π'. They are subsequently evaluated in *AccOrder1*. If $F(\Pi') < F(\Pi)$, d_2 is set to d_{min} and π_{orders} to π'_{orders} which means that π'_{orders} is used in the next iteration. Furthermore, in the case $F(\Pi') < F(\Pi^*)$, Π' is the new best found schedule Π^*. The order sequence π_{orders} is also set to π'_{orders} if $F(\Pi') = F(\Pi)$ but d_2 is increased by 1 in this case, if it has not already reached d_{max2}. Similar to IGA-String, π_{orders} can also be set to π'_{orders} if $F(\Pi') > F(\Pi)$, but only if $q \leq exp(-y \cdot \frac{d_{min}}{d_2} \cdot \frac{F(\Pi')-F(\Pi)}{F(\Pi)})$ holds, where q is a random number between 0 and 1, drawn each time. In addition, if $F(\Pi') > F(\Pi)$ and if d_2 is smaller than d_{max2}, d_2 is increased by 1. The pseudocode of this acceptance function can be found in Algorithm 15. As for IGA-String, IGA-Matrix1 terminates after the time limit is exceeded and the best found schedule Π^* and its total completion time are returned.

IGA-Matrix2 algorithm

Our next heuristic *IGA-Matrix2* is similar to IGA-Matrix1. However, this algorithm uses the function LocalSearch3 additionally after the initialization with MultipleNEH, and each time subsequent to MultipleInsertionOrder. By this function, the jobs of each order are not necessarily scheduled consecutively afterwards. The pseudocode of IGA-Matrix2 can be found in Algorithm 16.

The input of the function *LocalSearch3* is a schedule in which all jobs of an order are processed consecutively. Starting with the order in the last position of the order sequence, the first scheduled job of this order is swapped with the previously scheduled job. If the total completion time of the generated schedule is not lower than the total completion time of the former schedule, the former schedule is used for further processing. Otherwise, the new schedule is saved, and it is tested if a further improvement is possible by swapping the currently considered job with its new preceding job. This is repeated until a new obtained schedule does not have a lower total completion time or if the considered job passes the first scheduled job of the whole schedule. Subsequently, if swapping the job has improved the total completion time of the schedule before the job was stopped, the next job of this order in the corresponding job sequence $\pi_{jobs,i}$ is swapped forwards in the described way. This continues for an order until the first swap of a job does not improve the schedule, or all jobs of an order were considered. The procedure continues for all orders, except the first

Algorithm 16 General procedure of IGA-Matrix2

procedure IGA-Matrix2($d_{min}, d_{max1}, d_{max2}, y, z$)
  $\Pi, \pi_{orders}, \pi_{jobs} \leftarrow MultipleNEH()$
  $d_1, d_2 \leftarrow d_{max1}, d_{min}$
  $\Pi^* \leftarrow LocalSearch3(\Pi, \pi_{orders}, \pi_{jobs})$
  $\pi_{jobs} \leftarrow MultipleInsertionJobs(\pi_{jobs}, z, d_1, d_{min})$
  $\pi^*_{jobs} \leftarrow \pi_{jobs}$
  $\Pi, \pi_{orders} \leftarrow MultipleInsertionOrder(\pi_{orders}, \pi^*_{jobs}, z, d_2, d_{min})$
  $\Pi \leftarrow LocalSearch3(\Pi, \pi_{orders}, \pi^*_{jobs})$
  if $F(\Pi) < F(\Pi^*)$ **then**
    $\Pi^* \leftarrow \Pi$
  end if
  while time limit not exceeded **do**
    $\pi_{jobs:d}, \pi_{jobs:o-d} \leftarrow DestructionJobs(\pi_{jobs}, d_1)$
    $\pi'_{jobs} \leftarrow ConstructionJobs(\pi_{jobs:d}, \pi_{jobs:o-d})$
    $\pi'_{jobs} \leftarrow MultipleInsertionJobs(\pi'_{jobs}, z, d_1, d_{min})$
    $\pi^*_{jobs}, \pi_{jobs}, d_1 \leftarrow AccJobs(\pi^*_{jobs}, \pi'_{jobs}, \pi_{jobs}, d_1, d_{min}, y)$
    $\Pi \leftarrow \pi_{orders}, \pi^*_{jobs}$
    if $F(\Pi) < F(\Pi^*)$ **then**
      $\Pi^* \leftarrow \Pi$
    end if
    $\pi_{orders:d}, \pi_{orders:n-d} \leftarrow DestructionOrder(\pi_{orders}, d_2)$
    $\pi'_{orders} \leftarrow ConstructionOrder(\pi_{orders:d}, \pi_{orders:n-d}, \pi^*_{jobs})$
    $\Pi', \pi'_{orders} \leftarrow MultipleInsertionOrder(\pi'_{orders}, \pi^*_{jobs}, z, d_2, d_{min})$
    $\Pi'' \leftarrow LocalSearch3(\Pi', \pi'_{orders}, \pi^*_{jobs})$
    $\Pi^*, \pi_{orders}, d_2 \leftarrow AccOrder2(\Pi, \Pi^*, \Pi', \Pi'', \pi'_{orders}, \pi_{orders}, d_2, d_{min}, d_{max2}, y)$
  end while
  return $\Pi^*, F(\Pi^*)$
end procedure

Algorithm 17 LocalSearch3 in IGA-Matrix algorithms

procedure LocalSearch3(Π, π_{orders}, π_{jobs})
  $\Pi' \leftarrow \Pi$
  for $a \leftarrow n$ **to** 2 **do**
    $q, \delta_1, b \leftarrow \pi_{orders}(a), 1, 0$
    while $\delta_1 = 1$ **do**
      $\delta_1, b \leftarrow 0, b+1$
      $p, \delta_2 \leftarrow \pi_{jobs,q}(b), 1$
      while $\delta_2 = 1$ **do**
        $\delta_2 \leftarrow 0$
        $\Pi_{test} \leftarrow$ swap job (q, p) with the preceding job in Π'
        if $F(\Pi_{test}) < F(\Pi')$ **then**
          $\Pi' \leftarrow \Pi_{test}$
          if $b < o$ **then**
            $\delta_1 \leftarrow 1$
          end if
          if (q, p) is not in the first position of Π' now **then**
            $\delta_2 \leftarrow 1$
          end if
        end if
      end while
    end while
  end for
  return Π'
end procedure

Algorithm 18 Acceptance of order sequence in IGA-Matrix2

procedure AccOrder2($\Pi, \Pi^*, \Pi', \Pi'', \pi'_{orders}, \pi_{orders}, d_2, d_{min}, d_{max2}, y$)
  if $F(\Pi') < F(\Pi)$ **then**
    $\pi_{orders}, d_2 \leftarrow \pi'_{orders}, d_{min}$
  else if $F(\Pi') = F(\Pi)$ **then**
    $\pi_{orders} \leftarrow \pi'_{orders}$
    if $d_2 < d_{max2}$ **then**
      $d_2 \leftarrow d_2 + 1$
    end if
  else
    if $q \leq exp(-y \cdot \frac{d_{min}}{d_2} \cdot \frac{F(\Pi')-F(\Pi)}{F(\Pi)})$ **then**
      $\pi_{orders} \leftarrow \pi'_{orders}$
    end if
    if $d_2 < d_{max2}$ **then**
      $d_2 \leftarrow d_2 + 1$
    end if
  end if
  if $F(\Pi'') < F(\Pi^*)$ **then**
    $\Pi^* \leftarrow \Pi''$
  end if
  return Π^*, π_{orders}, d_2
end procedure

order in the order sequence π_{orders}. In Algorithm 17 the pseudocode of this function can be found.

Since we obtain a further schedule Π'' by LocalSearch3, there are more solutions to evaluate and hence, the second acceptance function of IGA-Matrix2 slightly differs from the second acceptance function of IGA-Matrix1. We call this acceptance function *AccOrder2* and the pseudocode of this function is given in Algorithm 18. As in the acceptance function of IGA-Matrix1, if $F(\Pi') < F(\Pi)$, d_2 is set to d_{min} and π_{orders} to π'_{orders}. Note that Π' is the schedule resulting from MultipleInsertionOrder and hence, is formed by the order sequence π'_{orders} and the job sequences π^*_{jobs}, while Π is formed by π_{orders} and π^*_{jobs}. The order sequence π_{orders} is also set to π'_{orders} if $F(\Pi') = F(\Pi)$ but d_2 is increased by 1 in this case if it has not already reached d_{max2}. If $F(\Pi') > F(\Pi)$, d_2 is increased by 1 if it has not already reached d_{max2}. Furthermore, π_{orders} is set to π'_{orders} if $q \leq exp(-y \cdot \frac{d_{min}}{d_2} \cdot \frac{F(\Pi')-F(\Pi)}{F(\Pi)})$ holds, where q is a random number between 0 and 1, drawn each time. In contrast to the acceptance function of IGA-Matrix1, the best found schedule Π^* is compared to Π''. Consequently, Π^* is set to Π'' if $F(\Pi'') < F(\Pi^*)$.

IGA-Matrix3 algorithm

Algorithm 19 General procedure of IGA-Matrix3

procedure IGA-MATRIX3($d_{min}, d_{max1}, d_{max2}, y, z$)
  $\Pi, \pi_{orders}, \pi_{jobs} \leftarrow MultipleNEH()$
  $d_1, d_2 \leftarrow d_{min}, d_{min}$
  $\Pi \leftarrow LocalSearch3(\Pi, \pi_{orders}, \pi_{jobs})$
  $\Pi^* \leftarrow \Pi$
  $\pi'_{jobs} \leftarrow MultipleInsertionJobs(\pi_{jobs}, z, d_1, d_{min})$
  $\Pi', \pi'_{orders} \leftarrow MultipleInsertionOrder(\pi_{orders}, \pi'_{jobs}, z, d_2, d_{min})$
  $\Pi' \leftarrow LocalSearch3(\Pi', \pi'_{orders}, \pi'_{jobs})$
  if $F(\Pi') \leq F(\Pi^*)$ **then**
    $\Pi^*, \Pi, \pi_{jobs}, \pi_{orders} \leftarrow \Pi', \Pi', \pi'_{jobs}, \pi'_{orders}$
  end if
  while time limit not exceeded **do**
    $\pi_{jobs:d}, \pi_{jobs:o-d} \leftarrow DestructionJobs(\pi_{jobs}, d_1)$
    $\pi'_{jobs} \leftarrow ConstructionJobs(\pi_{jobs:d}, \pi_{jobs:o-d})$
    $\pi'_{jobs} \leftarrow MultipleInsertionJobs(\pi'_{jobs}, z, d_1, d_{min})$
    $\pi_{orders:d}, \pi_{orders:n-d} \leftarrow DestructionOrder(\pi_{orders}, d_2)$
    $\pi'_{orders} \leftarrow ConstructionOrder(\pi_{orders:d}, \pi_{orders:n-d}, \pi'_{jobs})$
    $\Pi', \pi'_{orders} \leftarrow MultipleInsertionOrder(\pi'_{orders}, \pi'_{jobs}, z, d_2, d_{min})$
    $\Pi' \leftarrow LocalSearch3(\Pi', \pi'_{orders}, \pi'_{jobs})$
    $\Pi^*, \Pi, \pi_{orders}, \pi_{jobs}, d_1, d_2 \leftarrow Acc(\Pi^*, \Pi', \Pi, \pi'_{orders}, \pi_{orders}, \pi'_{jobs}, \pi_{jobs}, d_1, d_2, d_{min}, d_{max1}, d_{max2}, y)$
  end while
  return $\Pi^*, F(\Pi^*)$
end procedure

The pseudocode of our last heuristic *IGA-Matrix3* can be found in Algorithm 19. In contrast to IGA-Matrix1 and IGA-Matrix2, IGA-Matrix3 is not evaluating π_{jobs} independently from the generated schedule. This means, it is not checked whether new job sequences of the orders yield a lower makespan after MultipleInsertionJobs. Instead, they are directly used for restructuring the order sequence. Therefore, AccJobs is omitted, *Acc* is the only acceptance function in this algorithm, and no π^*_{jobs} is created. In addition, some minor changes were made compared to IGA-Matrix2 in order to adapt the algorithm to this strategy. After initializing a first solution, d_1 is set to d_{min} and not d_{max1}. In IGA-Matrix3, d_1 behaves similarly to d_2, i.e., set to d_{min} if a better solution was found in an iteration, or increased by 1 otherwise (if possible). Furthermore, the schedule Π which is compared in the acceptance function with the new created schedule of the iteration, is not generated during the iteration, i.e., not after AccJobs. Instead, it is the outputted schedule of the acceptance function of the previous iteration, or, to start the iterating part of the algorithm, one of the schedules generated by one of the previous LocalSearch3 functions. The part prior to the iterations is realized in the following way. After setting d_1 and d_2 as described, LocalSearch3 is applied to the schedule from the initialization function. The output is the updated schedule Π which is also marked as the first best found solution Π^*. Subsequently, MultipleInsertionJobs, MultipleInsertionOrder, and Localsearch3 are applied. The resulting schedule Π' is compared with Π^*, and hence, also with Π as it is the same as Π^*. If $F(\Pi') \leq F(\Pi^*)$, Π^* and Π are set to Π' and the job and order sequences for further proceeding (π_{jobs}, π_{orders}) are set to the corresponding π'_{jobs} and π'_{orders}. Afterwards, the iterating part begins. Here, as mentioned, IGA-Matrix3 is not using AccJobs and hence, the destruction of the order sequence directly follows MultipleInsertionJobs, i.e., after generating π'_{jobs} of the iteration. Furthermore, as π^*_{jobs} is not used in IGA-Matrix3, π'_{jobs} replaces it as input in the respective functions.

Algorithm 20 Acceptance in IGA-Matrix3

procedure ACC($\Pi^*, \Pi', \Pi, \pi'_{orders}, \pi_{orders}, \pi'_{jobs}, \pi_{jobs}, d_1, d_2, d_{min}, d_{max1}, d_{max2}, y$)
  if $F(\Pi') < F(\Pi)$ **then**
    $d_1, d_2, \pi_{orders}, \pi_{jobs}, \Pi \leftarrow d_{min}, d_{min}, \pi'_{orders}, \pi'_{jobs}, \Pi'$
    if $F(\Pi') < F(\Pi^*)$ **then**
      $\Pi^* \leftarrow \Pi'$
    end if
  else if $F(\Pi') = F(\Pi)$ **then**
    $\pi_{orders}, \pi_{jobs}, \Pi \leftarrow \pi'_{orders}, \pi'_{jobs}, \Pi'$
    if $d_1 < d_{max1}$ **then**
      $d_1 \leftarrow d_1 + 1$
    end if
    if $d_2 < d_{max2}$ **then**
      $d_2 \leftarrow d_2 + 1$
    end if
  else
    if $q \leq exp(-y \cdot \frac{d_{min}}{(d_1+d_2)\cdot 0.5} \cdot \frac{F(\Pi')-F(\Pi)}{F(\Pi)})$ **then**
      $\pi_{orders}, \pi_{jobs}, \Pi \leftarrow \pi'_{orders}, \pi'_{jobs}, \Pi'$
    end if
    if $d_1 < d_{max1}$ **then**
      $d_1 \leftarrow d_1 + 1$
    end if
    if $d_2 < d_{max2}$ **then**
      $d_2 \leftarrow d_2 + 1$
    end if
  end if
  return $\Pi^*, \Pi, \pi_{orders}, \pi_{jobs}, d_1, d_2$
end procedure

Next, we describe the acceptance function of IGA-Matrix3, called *Acc*, in detail. By this function, the best found solution Π^*, the destruction sizes d_1 and d_2 as well as Π, π_{orders} and π_{jobs} for the next iteration are selected. If a better solution was generated in the iteration, i.e., $F(\Pi') < F(\Pi)$, Π, π_{orders} and π_{jobs} are set to Π', π'_{orders} and π'_{jobs}, respectively. Furthermore, d_1 and d_2 are both set to d_{min}. In case $F(\Pi') < F(\Pi^*)$, Π' is saved as the best found solution Π^*. If $F(\Pi')$ equals $F(\Pi)$, Π, π_{orders} and π_{jobs} are also set to Π', π'_{orders} and π'_{jobs}, respectively, but d_1 and d_2 are increased by 1 if they have not already reached d_{max1} and d_{max2}, respectively. The same applies for the two destruction sizes if $F(\Pi')$ is greater than $F(\Pi)$. In this case, Π' could be accepted as new Π, and π_{orders} and π_{jobs} could be set to π'_{orders} and π'_{jobs}, respectively, but only if $q \leq exp(-y \cdot \frac{d_{min}}{(d_1+d_2)\cdot 0.5} \cdot \frac{F(\Pi')-F(\Pi)}{F(\Pi)})$ holds, where q is a random number between 0 and 1, drawn each time. The pseudocode of Acc is given in Algorithm 20.

5. Computational experiments

5.1. Experimental setting

We evaluate the developed IGAs in computational experiments. As there are no comparable state-of-the-art algorithms which take the specific problem structure into account, the four IGAs are compared with each other and the MINLP formulation from Section 3.2 solved

by the Gurobi solver. We generated two testbeds which we named *Small* testbed and *Big* testbed. The Small testbed includes problem instances with $n \in \{3,4,5\}$, $o \in \{2,3,4\}$ and $m \in \{2,3,6\}$, and the Big testbed problem instances with $n \in \{10,20,50\}$, $o \in \{2,5,10\}$ and $m \in \{2,3,6\}$. For each possible combination of n, m and o in the respective testbed, processing times of the jobs on the machines were randomly drawn from a uniform distribution $\mathcal{U}[1,100]$. We generated 10 problem instances per problem size for the parameter setting and 20 separate problem instances per problem size for the algorithm comparison.

We use the best found total completion time after given run times as the comparison criterion for the solution methods. The time limit of the IGAs is set to $n \cdot o \cdot \frac{m}{2} \cdot t\ sec$ with the time factor t set to $t = 0.12$. Consequently, the time limit for the smallest instances with $\{n,o,m\} = \{3,2,2\}$ is $0.72\ sec$ and for the largest problem instances with $\{n,o,m\} = \{50,10,6\}$ the time limit is $180\ sec$. As each of the IGAs contains random elements, each test instance was solved five times by each IGA. For the solver we set the time limit to $10,800\ sec$ ($= 3\ hours$). However, the solver solved each problem instance only one time.

The relative percentage deviation (RPD) was calculated for each run x by the formula:

$$RPD_x = \frac{F(\Pi_x) - F(\Pi_{best})}{F(\Pi_{best})} \cdot 100\%, \tag{9}$$

where $F(\Pi_x)$ is the total completion time of the best found solution in run x, and $F(\Pi_{best})$ the total completion time of the best solution found by any run of any tested solution method for the respective problem instance.

The solution methods have been implemented in Python 3.10.9 and the calculations run on an Intel(R) Xeon(R) CPU E5-2630 v2 processor with 2.60 GHz and 384 GB memory. We used *Gurobi* 10.0.1 as the solver for solving the MINLP from Section 3.2 with a maximum of 4 threads in parallel. The graphs were created with IBM SPSS Statistics version 29.

Statistical tests regarding the significance of results were conducted using IBM SPSS Statistics version 31. Note that prior the conduction of a statistical test, the RPDs of the five repetitions of an IGA for a problem instance were averaged.

5.2. Parameter setting

The four IGAs use multiple parameters, namely $d_{min}, d_{max}, d_{max1}, d_{max2}, y, z$. We follow the arguments from [11] and set $d_{min} = 1$, $d_{max} = \left\lfloor \frac{n \cdot o}{2} \right\rfloor$, $d_{max1} = \left\lfloor \frac{o}{2} \right\rfloor$ and $d_{max2} = \left\lfloor \frac{n}{2} \right\rfloor$. Note that the IGA-Matrix algorithms use d_{max1} and d_{max2} but not d_{max}, while IGA-String uses d_{max} but not d_{max1} and d_{max2}. The parameter d_{min} is used by all the IGAs.

The other parameters y and z were determined experimentally. As pointed out in [11], the ranges of the parameters are $1 \leq z \leq 2$ and $0 < y$. We performed some pre-experiments and chose $z \in \{1, 1.25, 1.5, 1.75, 2\}$ and $y \in \{5000, 10000, 15000, 20000\}$ for further investigation. Each possible parameter combination was considered for each IGA. As each of the 4 IGAs solved the 10 problem instances per problem size 5 times, and there are 54 different problem sizes and 20 different parameter combinations, $4 \cdot 10 \cdot 5 \cdot 54 \cdot 20 = 216,000$ runs were performed for the parameter setting.

The average RPDs for the different y-values and the average RPDs for the different z-values are given in Table 7 for each IGA. We chose the parameter value that led to the lowest average RPD for each IGA and parameter. The final selection is shown in Table 8.

5.3. Algorithm evaluation

In Table 9 we give the results for the Small testbed. For each problem size, it is indicated how many instances were solved confirmed optimally (see column # Slvd. Opt.), and for each IGA the percentage of runs were an optimal solution was found as well as the percentage of instances were the IGA found an optimal solution at least one time.

Table 7
Average RPDs in % of the investigated parameter values.

	Value	IGA-String	IGA-Matrix1	IGA-Matrix2	IGA-Matrix3
y	5000	0.5935	0.7862	0.6837	0.6104
	10 000	0.5921	0.7860	0.6825	0.6157
	15 000	0.5984	0.7848	0.6854	0.6209
	20 000	0.5909	0.7881	0.6852	0.6220
z	1	0.6152	0.7898	0.6828	0.6058
	1.25	0.6056	0.7861	0.6823	0.5996
	1.5	0.5949	0.7858	0.6834	0.6219
	1.75	0.5824	0.7836	0.6852	0.6152
	2	0.5705	0.7861	0.6874	0.6437

Table 8
Chosen parameter values for each IGA.

Parameter	IGA-String	IGA-Matrix1	IGA-Matrix2	IGA-Matrix3
y	20 000	15 000	10 000	5000
z	2	1.75	1.25	1.25

Both percentage values for the IGAs are relative to the instances which were solved confirmed optimally by the solver. Note that the IGAs solved each problem instance 5 times, while the solver solved each instance only one time. Furthermore, the RPDs (in %) of the solution methods are given.

As can be seen, the MINLP solver was able to find and confirm the optimal solution for each of the instances with $n = 3$. The same applies for the instances with $o = 2$. In total, 395 of the 540 instances, i.e., approx. 73%, were solved confirmed optimally. However, the solver could not confirm any optimal solution for the instances with $\{n,o\} = \{5,4\}$, and as it has an average RPD larger than zero, it is confirmed for multiple problem instances that they were not solved optimally by the MINLP solver.

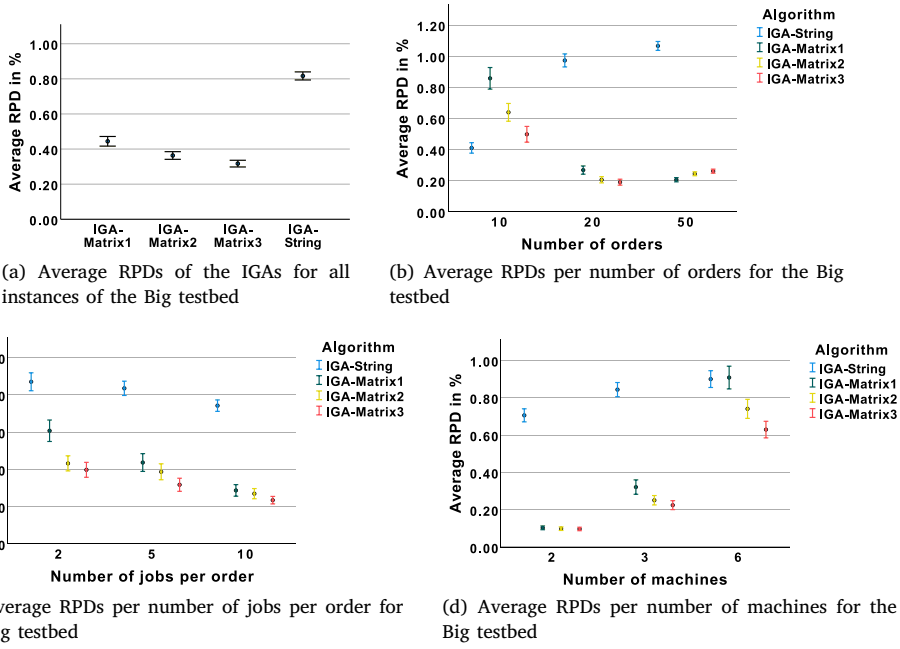
The best performing solution method (including the IGAs and the MINLP solver) for the Small testbed was IGA-String with an average RPD of 0.0575%. The result that IGA-String has a lower RPD than each other considered solution method for this testbed is statistically significant according to the one-sided Wilcoxon signed-rank test at a significance level of $\alpha = 0.05$. The heuristic could also find an optimal solution for 98.99% of the confirmed optimally solved instances and in 97.52% of the respective runs. But also the IGA-Matrix algorithms showed a good performance by finding an optimal solution for over 57% of the confirmed optimally solved instances. Furthermore, if an IGA-Matrix algorithm found an optimal solution for a confirmed optimally solved instance, it found an optimal solution in each run. And even though the IGA-Matrix algorithms did not find an optimal solution for about 40% of the confirmed optimally solved instances, the average RPD values are still only 1.06% (IGA-Matrix1), 0.95% (IGA-Matrix2), and 0.90% (IGA-Matrix3). We conclude by this that the four IGAs are appropriate and reliable solution methods for solving small sized problem instances of $Fm \rightarrow 0|prmu|\sum C_i$.

As expected, the MINLP solver performed worse for larger problem instances. This applies for each problem size defining parameter, i.e., number of orders, number of jobs per order, and number of machines. Besides not confirming an optimal solution for the problem instances with $\{n,o\} = \{5,4\}$ after 3 h, the lowest determined gap by the solver was 19.44%, and IGA-String and IGA-Matrix3 had a lower average RPD for these instances. Furthermore, the average gap determined by the solver was 27.95% and the highest gap 42.21% for these instances. Consequently, we did not include the MINLP solver for the Big testbed.

The average RPDs of the IGAs for the different problem sizes of the Big testbed are given in Table 10. The lowest average RPD for each problem size is written bold. If the lowest RPD-value is statistically significantly smaller than the RPD-value of each other IGA according to the one-sided Wilcoxon signed-rank test (significance level of $\alpha =$

Table 9
Results for the Small testbed.

n	o	m	MINLP		IGA-String			IGA-Matrix1			IGA-Matrix2			IGA-Matrix3		
			# Slvd. Opt.	RPD in %	% of Runs	% of Inst.	RPD in %	% of Runs	% of Inst.	RPD in %	% of Runs	% of Inst.	RPD in %	% of Runs	% of Inst.	RPD in %
		2	20	0.0000	100	100	0.0000	95	95	0.0581	95	95	0.0581	95	95	0.0581
	2	3	20	0.0000	100	100	0.0000	80	80	0.5397	80	80	0.4634	80	80	0.4634
		6	20	0.0000	100	100	0.0000	50	50	0.9748	65	65	0.7133	60	60	0.8607
		2	20	0.0000	100	100	0.0000	95	95	0.0588	95	95	0.0588	95	95	0.0588
3	3	3	20	0.0000	90	90	0.0626	45	45	1.2610	45	45	1.2450	45	45	1.2317
		6	20	0.0000	99	100	0.0022	35	35	1.4436	35	35	1.4436	35	35	1.4436
		2	20	0.0000	96	100	0.0209	90	90	0.4447	90	90	0.2741	90	90	0.2741
	4	3	20	0.0000	100	100	0.0000	55	55	0.8537	55	55	0.8537	60	60	0.7382
		6	20	0.0000	96	100	0.0674	10	10	1.9136	10	10	1.9150	15	15	1.8294
		2	20	0.0000	100	100	0.0000	90	90	0.1216	90	90	0.1216	90	90	0.1216
	2	3	20	0.0000	100	100	0.0000	45	45	0.8067	45	45	0.8067	45	45	0.8067
		6	20	0.0000	95	95	0.0149	50	50	1.1491	50	50	1.0656	50	50	1.0656
		2	20	0.0000	100	100	0.0000	75	75	0.1502	75	75	0.1502	75	75	0.1502
4	3	3	20	0.0000	98	100	0.0065	50	50	0.6130	50	50	0.6049	50	50	0.6049
		6	20	0.0000	90	95	0.0692	5	5	2.2912	5	5	2.0519	5	5	2.0155
		2	11	0.0000	100	100	0.0962	100	100	0.2657	100	100	0.2657	100	100	0.2657
	4	3	3	0.1535	93.33	100	0.1826	0	0	1.2062	0	0	1.1743	0	0	1.0984
		6	0	0.2308	–	–	0.0306	–	–	2.1703	–	–	2.1031	–	–	1.6967
		2	20	0.0000	96	100	0.0143	65	65	0.4848	70	70	0.3155	70	70	0.2720
	2	3	20	0.0000	99	100	0.0184	50	50	1.2990	55	55	0.7844	55	55	0.7844
		6	20	0.0000	99	100	0.0065	15	15	1.6245	20	20	1.2887	20	20	1.2887
		2	17	0.0000	96.47	100	0.0570	88.24	88.24	0.2927	88.24	88.24	0.1566	88.24	88.24	0.1566
5	3	3	4	0.0000	85	100	0.0716	25	25	1.3715	25	25	1.3027	25	25	1.3402
		6	0	0.4431	–	–	0.1246	–	–	2.7055	–	–	2.4401	–	–	2.4044
		2	0	0.3178	–	–	0.1519	–	–	0.2663	–	–	0.2066	–	–	0.2066
	4	3	0	1.2104	–	–	0.2416	–	–	1.2704	–	–	1.1790	–	–	0.9828
		6	0	1.7966	–	–	0.3141	–	–	3.1142	–	–	2.4965	–	–	2.0363
Total			395	0.1538	97.52	98.99	0.0575	57.47	57.47	1.0649	58.99	58.99	0.9459	59.24	59.24	0.8983



(a) Average RPDs of the IGAs for all instances of the Big testbed

(b) Average RPDs per number of orders for the Big testbed

(c) Average RPDs per number of jobs per order for the Big testbed

(d) Average RPDs per number of machines for the Big testbed

Fig. 4. Results of the IGAs for the Big testbed.

0.05), we marked the corresponding value with †. Furthermore, Fig. 4(a) shows the average RPD of each IGA across all problem instances of the Big testbed with 95% confidence intervals. As for the Small testbed, IGA-Matrix3 had a lower average RPD than IGA-Matrix1 and IGA-Matrix2 when the complete Big testbed is considered. This result is statistically significant according to the one-sided Wilcoxon signed-rank test at a significance level of $\alpha = 0.05$. From this we conclude that it is generally advisable to use only one acceptance function at the end of each iteration, so that the job sequences of each order and the order sequence are not evaluated separately. In contrast to the results from the Small testbed, IGA-String had the highest average RPD for the Big testbed. The result that IGA-String has a higher RPD than each other IGA for the big testbed is statistically significant according to the one-sided Wilcoxon signed-rank test at a significance level of $\alpha = 0.05$.

An explanation for this is indicated by Fig. 4(b). In the respective figure, the average RPDs with a 95% confidence interval for the IGAs per number of orders is illustrated for the Big testbed. It can be seen that IGA-String had the best performance for $n = 10$. However, for higher

Table 10
Average RPDs in % of the IGAs for different problem sizes of the Big testbed.

n	o	m	IGA-String	IGA-Matrix1	IGA-Matrix2	IGA-Matrix3
		2	**0.1522**	0.2177	0.1822	0.1847
	2	3	**0.2513**†	1.1948	0.6424	0.6285
		6	**0.3300**†	1.8962	1.2979	1.2275
		2	0.3642	0.1090	**0.0740**	0.0763
10	5	3	0.7840	0.5625	0.3967	**0.2552**†
		6	**0.3991**†	2.0614	1.7858	1.2659
		2	0.3056	0.1427	0.0436	**0.0268**†
	10	3	0.6528	0.1714	0.1237	**0.0774**†
		6	**0.4579**	1.3658	1.2084	0.7434
		2	1.0672	0.0402	0.0410	**0.0351**
	2	3	1.0767	0.2180	0.1533	**0.1315**
		6	1.2952	0.9452	0.6124	**0.5455**†
		2	1.0989	0.0265	**0.0209**	0.0221
20	5	3	0.9316	0.1216	0.1245	**0.0989**
		6	0.6809	0.5025	0.3823	**0.3520**
		2	0.7309	0.0567	**0.0367**	0.0430
	10	3	0.9124	**0.1155**	0.1157	0.1387
		6	0.9603	0.3900	0.3681	**0.3518**
		2	0.8218	**0.1401**†	0.1795	0.1795
	2	3	1.1086	0.2743	0.2985	**0.2404**
		6	1.7349	0.5453	0.4856	**0.4106**†
		2	1.0165	**0.1206**†	0.1717	0.1599
50	5	3	1.0541	**0.1492**†	0.2217	0.2421
		6	1.1928	**0.2817**	0.3088	0.3883
		2	0.7949	**0.0732**†	0.1363	0.1461
	10	3	0.8215	**0.0862**†	0.1814	0.2088
		6	1.0488	**0.1865**	0.2164	0.3800
Total			0.8165	0.4443	0.3633	**0.3170**

number of orders, the performance of IGA-String was notably worse compared to the IGA-Matrix algorithms. We assume that this is due to the higher computational effort per iteration that IGA-String has, since this algorithm modifies the sequence of all jobs of all orders and not partial sequences.

Furthermore, IGA-Matrix1 had the best performance for the highest number of orders, i.e., $n = 50$. The result that IGA-Matrix1 has a lower RPD than each other IGA for $n = 50$ is statistically significant according to the one-sided Wilcoxon signed-rank test at a significance level of $\alpha = 0.05$. This finding can also be explained by the differences in computational effort per iteration as IGA-Matrix1 does not use LocalSearch3 and hence, can perform more iterations until the given time limit is reached. We explain the result for the different number of jobs per order, see Fig. 4(c), in the same way. Here, the gap between IGA-Matrix1 and the other IGA-Matrix algorithms narrows for a larger number of jobs per order. However, for the highest number of jobs per order (o=10), IGA-Matrix3 still had the lowest average RPD.

By Fig. 4(d), we give the average RPDs with a 95% confidence interval for the IGAs per number of machines for the Big testbed. For each IGA, the average RPD and the 95% confidence interval increase for higher m. Our conclusion is that for a higher number of machines, the solution quality of the IGAs becomes more volatile. Furthermore, we make the following observation when we compare the IGAs with each other. The gap between IGA-Matrix1 and IGA-Matrix2, the gap between IGA-Matrix1 and IGA-Matrix3, as well as the gap between IGA-Matrix2 and IGA-Matrix3 widens for higher numbers of machines. Furthermore, the gaps between IGA-String and each IGA-Matrix algorithm are noticeably smaller for $m = 6$ than for $m = 2$, and for $m = 6$, IGA-String performs even slightly better than IGA-Matrix1.

We conclude by this that IGA-String performs relatively better for instances with a high number of machines. The same applies for IGA-Matrix3 when it is compared with the other IGA-Matrix algorithms, and for IGA-Matrix2 when it is compared with IGA-Matrix1. We explain this result in the following way. For a higher number of machines, it becomes more important that two consecutively scheduled jobs fit to each other, i.e., that the forced idle times and waiting times between the two jobs become low, as there are more machines where forced idle times and waiting times can occur. Consequently, it is advisable to check more combinations of consecutively scheduled jobs. This can be achieved by also considering schedules where the jobs of an order are not scheduled consecutively, since this results in checking more jobs of different orders for consecutive scheduling. In contrast to IGA-Matrix2 and IGA-Matrix3, IGA-Matrix1 only considers schedules where the jobs of an order are scheduled consecutively which explains the corresponding result. The difference between IGA-Matrix2 and IGA-Matrix3 is that IGA-Matrix3 does not evaluate the job and order sequences independently of each other. We assume that this leads to a stronger focus on the fit of the consecutively scheduled jobs in the whole schedule instead of focusing on the fit of the jobs within the own order. Meanwhile, IGA-String uses only one job sequence of all jobs of all orders after the initialization function, and not the separate job sequences of each order. By this it is more likely that jobs of different orders are scheduled consecutively which leads to a broader search for good fitting job combinations.

6. Conclusion

In this paper, we studied the minimization of the total completion time of the COSP in the permutation flow shop environment. We presented important problem properties and used these to give a MINLP formulation of the problem. Furthermore, we developed four heuristics for the problem which are based on the IGA. One of the heuristics, IGA-String, modifies directly the job sequence of all jobs of all orders after the initialization function, while the other three algorithms, called IGA-Matrix algorithms, first modify the job sequences of each order separately and afterwards the order sequence. IGA-Matrix2 and IGA-Matrix3 additionally have the LocalSearch3 function by which the jobs of an order are scheduled actively between jobs of another order. Furthermore, IGA-Matrix1 and IGA-Matrix2 use two acceptance functions for evaluating the job sequences of the single orders independent from the order sequence, while IGA-Matrix3 uses only one acceptance function.

We compared the developed IGAs with each other and the MINLP solution from the Gurobi solver. For the instances which where confirmed optimally solved by the solver, IGA-String was able to find an optimal solution in 97.52% of the runs. Each of the IGA-Matrix algorithms found an optimal solution in over 57% of the runs for these instances. This shows that the developed algorithms are reliable and appropriate solution methods for the considered problem.

For larger problem instances, IGA-Matrix3 showed the best performance, especially because IGA-String performed worse for a higher number of orders, probably due to the high computational effort per iteration. As IGA-Matrix3 performed better than the other two IGA-Matrix algorithms for both, the Small and the Big testbed, we furthermore conclude that the job sequences of the single orders and the order sequence shall not be evaluated independent of each other.

Future research may investigate the flow shop environment for the COSP with further objective functions, e.g., the total tardiness, and machine specifications, e.g., with limited intermediate storage. Furthermore, as literature about the COSP continuously grows, this problem class deserves a more general view by a systematic literature review in which the key results of the different problem configurations are summarized.

CRediT authorship contribution statement

Julius Hoffmann: Writing – original draft, Visualization, Validation, Supervision, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Janis S. Neufeld:** Writing – review & editing, Supervision, Methodology, Conceptualization. **Udo Buscher:** Writing – review & editing, Supervision, Resources, Methodology, Conceptualization.

Declaration of competing interest

The authors report there are no competing interests to declare.

Data availability

The problem instances used for the computational experiments can be found at https://github.com/Julius2627/COSP_Flow_Shop.

References

[1] Framinan JM, Perez-Gonzalez P. New approximate algorithms for the customer order scheduling problem with total completion time objective. Comput Oper Res 2017;78:181–92. http://dx.doi.org/10.1016/j.cor.2016.09.010.

[2] Sung CS, Yoon SH. Minimizing total weighted completion time at a pre-assembly stage composed of two feeding machines. Int J Prod Econ 1998;54(3):247–55. http://dx.doi.org/10.1016/S0925-5273(97)00151-5.

[3] Framinan JM, Perez-Gonzalez P, Fernandez-Viagas V. Deterministic assembly scheduling problems: A review and classification of concurrent-type scheduling models and solution procedures. European J Oper Res 2019;273(2):401–17. http://dx.doi.org/10.1016/j.ejor.2018.04.033.

[4] Zhao Z, Zhou M, Liu S. Iterated greedy algorithms for flow-shop scheduling problems: A tutorial. IEEE Trans Autom Sci Eng 2022;19(3):1941–59. http://dx.doi.org/10.1109/TASE.2021.3062994.

[5] Wagneur E, Sriskandarajah C. Openshops with jobs overlap. European J Oper Res 1993;71(3):366–78. http://dx.doi.org/10.1016/0377-2217(93)90347-P.

[6] Leung JY-T, Li H, Pinedo M, Sriskandarajah C. Open shops with jobs overlap—-revisited. European J Oper Res 2005;163(2):569–71. http://dx.doi.org/10.1016/j.ejor.2003.11.023.

[7] Roemer TA. A note on the complexity of the concurrent open shop problem. J Sched 2006;9(4):389–96. http://dx.doi.org/10.1007/s10951-006-7042-y.

[8] Leung JY-T, Li H, Pinedo M. Order scheduling in an environment with dedicated resources in parallel. J Sched 2005;8(5):355–86. http://dx.doi.org/10.1007/s10951-005-2860-x.

[9] Yang J, Posner ME. Scheduling parallel machines for the customer order problem. J Sched 2005;8(1):49–74. http://dx.doi.org/10.1007/s10951-005-5315-5.

[10] Xu X, Ma Y, Zhou Z, Zhao Y. Customer order scheduling on unrelated parallel machines to minimize total completion time. IEEE Trans Autom Sci Eng 2015;12(1):244–57. http://dx.doi.org/10.1109/TASE.2013.2291899.

[11] Hoffmann J, Neufeld JS, Buscher U. Iterated greedy algorithms for customer order scheduling with dedicated machines. IFAC-PapersOnLine 2022;55(10):1594–9. http://dx.doi.org/10.1016/j.ifacol.2022.09.618.

[12] Lee IS. Minimizing total tardiness for the order scheduling problem. Int J Prod Econ 2013;144(1):128–34. http://dx.doi.org/10.1016/j.ijpe.2013.01.025.

[13] Framinan JM, Perez-Gonzalez P. Order scheduling with tardiness objective: Improved approximate solutions. European J Oper Res 2018;266(3):840–50. http://dx.doi.org/10.1016/j.ejor.2017.10.064.

[14] Braga-Santos SA, Barroso GC, Prata BdA. A size-reduction algorithm for the order scheduling problem with total tardiness minimization. J Proj Manag 2022;7(3):167–76. http://dx.doi.org/10.5267/j.jpm.2022.1.001.

[15] Lin W-C, Xu J, Bai D, Chung I-H, Liu S-C, Wu C-C. Artificial bee colony algorithms for the order scheduling with release dates. Soft Comput 2019;23(18):8677–88. http://dx.doi.org/10.1007/s00500-018-3466-5.

[16] Prata BDA, Rodrigues CD, Framinan JM. Customer order scheduling problem to minimize makespan with sequence-dependent setup times. Comput Ind Eng 2021;151:106962. http://dx.doi.org/10.1016/j.cie.2020.106962.

[17] Prata BDA, Rodrigues CD, Framinan JM. A differential evolution algorithm for the customer order scheduling problem with sequence-dependent setup times. Expert Syst Appl 2022;189:116097. http://dx.doi.org/10.1016/j.eswa.2021.116097.

[18] Shi Z, Huang Z, Shi L. Customer order scheduling on batch processing machines with incompatible job families. Int J Prod Res 2018;56(1–2):795–808. http://dx.doi.org/10.1080/00207543.2017.1401247.

[19] Li L-Y, Xu J-Y, Cheng S-R, Zhang X, Lin W-C, Lin J-C, Wu Z-L, Wu C-C. A genetic hyper-heuristic for an order scheduling problem with two scenario-dependent parameters in a parallel-machine environment. Mathematics 2022;10(21):4146. http://dx.doi.org/10.3390/math10214146.

[20] Wu C-C, Gupta JND, Lin W-C, Cheng S-R, Chiu Y-L, Chen J-H, Lee L-Y. Robust scheduling of two-agent customer orders with scenario-dependent component processing times and release dates. Mathematics 2022;10(9):1545. http://dx.doi.org/10.3390/math10091545.

[21] Zhao Y, Kong X, Xu X, Xu E. Resource-controlled stochastic customer order scheduling via particle swarm optimization with bound information. Ind Manag Data Syst 2022;122(8):1882–908. http://dx.doi.org/10.1108/IMDS-02-2022-0105.

[22] Lin W-C, Zhang X, Liu X, Hu K-X, Cheng S-R, Azzouz A, Wu C-C. Sequencing single machine multiple-class customer order jobs using heuristics and improved simulated annealing algorithms. RAIRO - Oper Res 2023;57(3):1417–41. http://dx.doi.org/10.1051/ro/2023056.

[23] Liu G, Xie Y, Wang H. Customer order scheduling on a serial-batch machine in precast bridge construction. Comput Oper Res 2025;173:106871. http://dx.doi.org/10.1016/j.cor.2024.106871.

[24] Hsu S-Y, Liu C-H. Improving the delivery efficiency of the customer order scheduling problem in a job shop. Comput Ind Eng 2009;57(3):856–66. http://dx.doi.org/10.1016/j.cie.2009.02.015.

[25] Yang J. Customer order scheduling in a two machine flowshop. Manag Sci Financ Eng 2011;17(1):95–116.

[26] Chen J, Wang M, Kong XTR, Huang GQ, Dai Q, Shi G. Manufacturing synchronization in a hybrid flowshop with dynamic order arrivals. J Intell Manuf 2019;30(7):2659–68. http://dx.doi.org/10.1007/s10845-017-1295-5.

[27] Meng T, Pan Q-K, Wang L. A distributed permutation flowshop scheduling problem with the customer order constraint. Knowl-Based Syst 2019;184:104894. http://dx.doi.org/10.1016/j.knosys.2019.104894.

[28] Li W, Chen X, Li J, Sang H, Han Y, Du S. An improved iterated greedy algorithm for distributed robotic flowshop scheduling with order constraints. Comput Ind Eng 2022;164:107907. http://dx.doi.org/10.1016/j.cie.2021.107907.

[29] Yang S, Xu Z. The distributed assembly permutation flowshop scheduling problem with flexible assembly and batch delivery. Int J Prod Res 2021;59(13):4053–71. http://dx.doi.org/10.1080/00207543.2020.1757174.

[30] Zhou X, Ling G, Yu J, Zhou T, Wang R. Balanced multi-objective evolution algorithm for unmanned systems project scheduling with preventive maintenance and order grouping constraints. Expert Syst Appl 2026;299:130006. http://dx.doi.org/10.1016/j.eswa.2025.130006.

[31] Erel E, Ghosh JB. Customer order scheduling on a single machine with family setup times: Complexity and algorithms. Appl Math Comput 2007;185(1):11–8. http://dx.doi.org/10.1016/j.amc.2006.06.086.

[32] Hoffmann J, Neufeld JS, Buscher U. Minimizing the earliness–tardiness for the customer order scheduling problem in a dedicated machine environment. J Sched 2024;27(6):525–43. http://dx.doi.org/10.1007/s10951-024-00814-z.

[33] Wu C-C, Liu S-C, Lin T-Y, Yang T-H, Chung I-H, Lin W-C. Bicriterion total flowtime and maximum tardiness minimization for an order scheduling problem. Comput Ind Eng 2018;117:152–63. http://dx.doi.org/10.1016/j.cie.2018.01.011.

[34] Wu C-C, Yang T-H, Zhang X, Kang C-C, Chung I-H, Lin W-C. Using heuristic and iterative greedy algorithms for the total weighted completion time order scheduling with release times. Swarm Evol Comput 2019;44:913–26. http://dx.doi.org/10.1016/j.swevo.2018.10.003.

[35] Çetinkaya FC, Yozgat GB. Customer order scheduling with job-based processing and lot streaming in a two-machine flow shop. Int J Industiral Eng Prod Res 2022;33(2). http://dx.doi.org/10.22068/ijiepr.33.2.11.

[36] Xiong F, Chen S, Ma Z, Li L. Approximate model and algorithms for precast supply chain scheduling problem with time-dependent transportation times. Int J Prod Res 2023;61(7):2057–85. http://dx.doi.org/10.1080/00207543.2022.2057254.

[37] Pinedo ML. Deterministic models: Preliminaries. In: Scheduling. 3rd ed. New York, NY: Springer New York; 2008, p. 13–33. http://dx.doi.org/10.1007/978-0-387-78935-4_2.

[38] Garey MR, Johnson DS, Sethi R. The complexity of flowshop and jobshop scheduling. Math Oper Res 1976;1(2):117–29. http://dx.doi.org/10.1287/moor.1.2.117.

[39] Conway RW, Maxwell WL, Miller LW. Theory of scheduling. Reading, Mass.: Addison-Wesley; 1967.

[40] Nawaz M, Enscore Jr. EE, Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 1983;11(1):91–5. http://dx.doi.org/10.1016/0305-0483(83)90088-9.

[41] Neufeld JS, Gupta JND, Buscher U. Minimising makespan in flowshop group scheduling with sequence-dependent family set-up times using inserted idle times. Int J Prod Res 2015;53(6):1791–806. http://dx.doi.org/10.1080/00207543.2014.961209.

[42] Neufeld JS, Gupta JND, Buscher U. A comprehensive review of flowshop group scheduling literature. Comput Oper Res 2016;70:56–74. http://dx.doi.org/10.1016/j.cor.2015.12.006.