Journal of Scheduling
https://doi.org/10.1007/s10951-024-00814-z

Minimizing the earliness–tardiness for the customer order scheduling problem in a dedicated machine environment

Julius Hoffmann[1,2] · Janis S. Neufeld[1,3] · Udo Buscher[1]

Accepted: 10 June 2024

Abstract
The customer order scheduling problem has garnered considerable attention in the recent scheduling literature. It is assumed that each of several customer orders consists of several jobs, and each customer order is completed only if each job of the order is completed. In this paper, we consider the customer order scheduling problem in a machine environment where each customer places exactly one job on each machine. The objective is to minimize the earliness–tardiness, where tardiness is defined as the time an order is finished past its due date, and earliness is the time a job is finished before its due date or the completion time of the corresponding order, whichever is later. Even though the earliness–tardiness criterion is an important objective for just-in-time production, this problem has not been studied in the context of the customer order scheduling problem. We provide a mixed-integer linear programming (MILP) formulation for this problem and derive multiple problem properties. Furthermore, we develop six different heuristics for this problem configuration. They follow the structure of the iterated greedy algorithm and additionally use a refinement function in which they differ. In a computational experiment, the algorithms were compared with each other and outperformed a solver solution of the MILP, which proves their ability to efficiently solve the problem configuration.

Keywords Machine scheduling · Customer order scheduling · Iterated greedy algorithm · Metaheuristics · Earliness–tardiness · Dedicated machines

1 Introduction

Classical scheduling problems involve scheduling a set of jobs that are considered as single entities to optimize a job-related objective. Therefore, it is assumed that the resulting products are shipped individually after processing. However, in many real-world scenarios, customers order multiple products. Shipping the products of a customer order separately results in higher transportation costs and distribution execution time. In addition, storage costs for the customer increase when parts of an order are delivered at different times but are all needed together for further processing (Framinan & Perez-Gonzalez, 2017).

This issue is addressed by the customer order scheduling problem (COSP). In contrast to classical scheduling problems, the COSP assumes that each order consists of multiple jobs and is finished when all jobs of the order have been processed. Consequently, the completion time of an order corresponds to the completion time of the last finished job of the order. The COSP has been investigated for different objectives and machine environments (see Sect. 2 for examples). We consider the so-called dedicated machine environment, where each order has exactly one job on each machine. This is equivalent to the assumption that customers can order different product types, and each product type can only be processed on a dedicated machine.

✉ Julius Hoffmann
julius.hoffmann@kit.edu

Janis S. Neufeld
janis.neufeld@ovgu.de

Udo Buscher
udo.buscher@tu-dresden.de

1 Faculty of Business and Economics, Chair for Industrial Management, TU Dresden, Helmholtzstr. 10, 01069 Dresden, Germany

2 Institute for Operations Research, Discrete Optimization and Logistics, Karlsruhe Institute of Technology, Kaiserstr. 89, 76133 Karlsruhe, Germany

3 Chair of Operations Management, Otto-von-Guericke-Universität Magdeburg, Universitätsplatz 2, 39106 Magdeburg, Germany

Published online: 13 October 2024

A well-known objective of scheduling problems is the minimization of earliness–tardiness. On the one hand, minimizing tardiness leads to higher customer satisfaction and prevents penalty costs. On the other hand, minimizing premature completion of products decreases warehousing costs (Kedad-Sidhoum & Sourd, 2010). Moreover, the time between the completion of a product and its shipment becomes critical when handling perishable goods, which can be found in the food and pharmaceutical industries.

In classical scheduling problems, the earliness of a job is defined as the time that a job is finished before its due date (or zero if the job is finished after the due date), and the tardiness as the time a job is finished past its due date (or zero if the job is finished before the due date) (Kramer & Subramanian, 2019). Both assume that the job is shipped at the due date or the completion time of the job, whichever is later. To take into account the specifics of the COSP, the definition of earliness–tardiness must be adjusted to the COSP in order to continue to minimize storage costs and maximize customer satisfaction at the same time. Therefore, the tardiness of an order is defined as the time an order is finished past the due date of the order (or zero if the order is finished before the due date), and the earliness of a job as the time that a job is finished before the due date or the completion time of the order, whichever is later. The definition of earliness is based on the assumption that a finished product must be stored until its shipment date.

To the best of our knowledge, the earliness–tardiness objective has not yet been studied in the context of the COSP. In this work, we investigate this objective in the dedicated machine environment. The contributions of this paper are as follows:

- We derive multiple properties of the problem configuration which are necessary for our problem formulation and are used in the design of solution algorithms.
- We present a mixed-integer linear programming (MILP) formulation for the problem.
- We develop six heuristics which are proved to efficiently solve large instances in a computational experiment.

For this purpose, the remainder of this article is structured as follows. In Sect. 2, we present an overview of the literature related to the COSP and minimization of the earliness–tardiness. Section 3 presents a formal description of the problem as well as a MILP formulation and problem properties. Based on these properties, we describe developed metaheuristics in Sect. 4, which are tested in a computational experiment in Sect. 5. In Sect. 6, we present conclusions based on the results of this study and provide an outlook for future research possibilities.

2 Literature review

In this section, we discuss the relevant literature regarding the COSP and respective solution approaches, the dedicated machine environment, and the objective of minimizing earliness–tardiness.

The COSP is a well-known scheduling problem and has been studied for various problem settings. For example, objectives such as minimizing the total weighted completion time (Leung et al., 2007; Shi et al., 2018) and the number of tardy orders (Wu et al., 2018) have been addressed. In addition, different machine environments have been considered, such as the identical machine environment (Leung et al., 2008a, b), the batch machine environment (Shi et al., 2018), and the job shop environment (Liu, 2009). The more recent literature about the COSP includes Li et al. (2022) and Wu et al. (2022), who focus on robust solution approaches for the COSP with scenario-dependent problem parameters, as well as Li et al. (2023), who investigate a tri-criteria COSP in the single-machine environment with job classes by studying multiple heuristics and a branch-and-bound algorithm. A brief overview of variants of the COSP is presented in Framinan et al. (2019). The authors also point out that the COSP is a special case of the assembly scheduling problem and introduce a notation for configurations of the assembly scheduling problem, which we adopt in the following.

Multiple problem configurations of the COSP have been shown to be NP-hard. For example, Roemer (2006) proved that $\left(DPm \rightarrow 0||\sum C_i\right)$ is NP-hard, even for the two-machine case. Consequently, various heuristics have been established to solve variants of the COSP. Simulated annealing algorithms were applied by Xu et al. (2016) and Kung et al. (2018) and tabu search algorithms by Leung et al. (2005) and Li and Yoon (2015) to solve different types of COSPs. Furthermore, Hazür et al. (2008) performed a comparative study of the performance of four different metaheuristics applied to the COSP in a single-machine environment. A metaheuristic that has been used in more recent COSP studies is the iterated greedy algorithm (IGA); see Wu et al. (2019), Wu et al. (2021), and Hoffmann et al. (2022) for examples.

One of the most widely investigated machine environments of the COSP is the dedicated machine environment. Much attention has recently been focused on minimizing the total tardiness in this machine setting. Braga-Santos et al. (2022) presented a size reduction algorithm for the standard configuration $\left(DPm \rightarrow 0||\sum T_i\right)$, while in de Abreu et al. (2022), a biased random key genetic algorithm is developed and missing operations of single orders are taken into account $\left(DPm \rightarrow 0|missing|\sum T_i\right)$. Furthermore, in Antonioli et al. (2022), the problem configuration is extended with sequence-dependent setup times $\left(DPm \rightarrow 0|ST_{sd}|\sum T_i\right)$.

To the best of our knowledge, the problem of minimizing total tardiness in the dedicated machine environment was first addressed by Lee (2013). Important properties of the problem configuration were presented in that paper. It was noted that $DPm \rightarrow 0||\sum T_i$ is NP-hard, even for the two-machine case. It was further shown that there always exists an optimal schedule without idle times and where each machine processes the orders in the same sequence. The latter property is a specification of a lemma from Leung et al. (2005) and applies to different objective functions in the dedicated machine environment. The lemma from that paper states that for the problem configuration $DPm \rightarrow 0||\sum f_i(C_i)$, where $f_i(C_i)$ increases in C_i for each i, there exists an optimal schedule for each problem instance in which each machine processes the orders in the same sequence.

The objective of minimizing the earliness–tardiness has been addressed in various studies for different problem settings. In Polyakovskiy and M'Hallah (2014), minimizing the weighted earliness–tardiness was studied in a nonidentical parallel-machine environment. The weights for the earliness and tardiness costs are job-dependent and can differ between earliness and tardiness. The same objective was studied by Yousefi and Yusuff (2013) for the single-machine environment, where all jobs share a common due date. The common due date property was also considered for minimizing the weighted earliness–tardiness in the unrelated parallel-machine environment in Bank and Werner (2001). An interesting property of job scheduling with a common due date for minimizing earliness–tardiness in specific machine environments is that there exists an optimal schedule without idle times inserted between the scheduled jobs. This applies, for example, in the single-machine environment, but also in the parallel-machine environment with unrelated machines when there are no further restrictions beyond the common due date (Bank & Werner, 2001). Note that in the COSP, the jobs of the same order also have a common due date. A recent bibliography of earliness–tardiness problems in general was given in Kramer and Subramanian (2019).

Even though earliness–tardiness has not yet been studied for the COSP in the way it is defined here, there are papers that address problems with similar intentions. Minimizing the sum of the longest waiting durations, which is defined as the difference between the completion times of the first and last finished job of an order, was studied in Li and Yoon (2015), Daoud et al. (2018), Li et al. (2018), and Li et al. (2023). In Liu (2009), the sum of the absolute lateness values, which are defined as the deviation of the completion time of an order from its due date, was minimized in the context of the COSP. Meanwhile, Gerstl and Mosheiov (2012) addressed minimizing earliness–tardiness in the parallel uniform machine environment, where jobs belong to classes and the jobs of a class have the same due date. However, this study was not within the scope of the COSP.

Table 1 Notation

I	Set of orders
J	Set of jobs
IT	Set of idle times
n	Number of customer orders
m	Number of machines
p_{ij}	Processing time of job j in order i
d_i	Due date of order i
x_{ijh}	Assignment of job j of order i to position h
z_{ijh}	Completion time of job j of order i, when it is assigned to position h
C_i	Completion time of order i
C_{ij}	Completion time of job j in order i
T_i	Tardiness of order i
E_{ij}	Earliness of job j in order i
M	Large number
n_g	Number of jobs in group g
v_g	Number of jobs in group g with an earliness of 0
it_{ij}	Idle time before job j from order i
ds	Destruction size
ds_{min}	Minimum destruction size
ds_{max}	Maximum destruction size
y, z	Algorithm parameters
π_{or}	Sequence of orders
π_{ma}	Set of sequences
$\pi_{ma,a}$	Sequence of jobs on machine a
Π	Feasible schedule
Π^*	Best found feasible schedule
$F(\cdot)$	Objective function value
TF	Tardiness factor
RDD	Range of due dates

3 Problem description

3.1 Problem definition

Next, we define the configuration of the studied problem. The notations for variables and parameters used in this paper are shown in Table 1. There are n customers who place exactly one job on each of m dedicated machines. The numbering of a job of an order corresponds to the number of the machine. Therefore, job j of order i is processed on machine j and has a processing time p_{ij} and, based on the (partial) schedule, a completion time C_{ij}. We denote this job with (i, j), or with the order number if the considered machine is obvious. It is possible to insert an idle time it_{ij} before the job (i, j). An order is completed the moment all jobs have finished, and hence, the completion time of an order i is defined as $C_i = \max_{1\leq j\leq m}\{C_{ij}\}$. Each order has a due date d_i. The

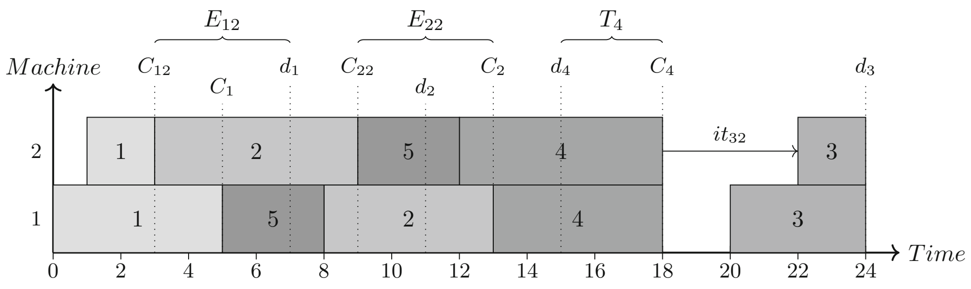

Fig. 1 Gantt chart of an exemplary schedule

tardiness of an order is defined by $T_i = \max\{0, C_i - d_i\}$. We define the earliness of a job as $E_{ij} = \max\{d_i, C_i\} - C_{ij}$. Note that $E_{ij} \geq 0$, since $C_i \geq C_{ij}$. The objective is to minimize the earliness–tardiness that we define as $\sum_{i=1}^{n}\sum_{j=1}^{m}\left(E_{ij} + T_i\right) = \sum_{i=1}^{n}\left(\sum_{j=1}^{m}\left(E_{ij}\right) + m \cdot T_i\right)$. The problem configuration has the notation $DPm \rightarrow 0||\sum\left(\sum E_{ij} + mT_i\right)$ when preemption is not allowed, and $DPm \rightarrow 0|prmp|\sum\left(\sum E_{ij} + mT_i\right)$ for the preemptive case.

For the purpose of clarity, an exemplary schedule for the non-preemptive case with five orders on two machines is presented in Fig. 1. The number in each box represents the order number. Each order places one job on each machine and is completed when all jobs have finished, as can be seen by C_1. Before each job, an idle time can be inserted, as is depicted for job 2 of order 3 (see it_{32}). The earliness of two jobs is given in the Gantt chart. The earliness E_{12} is the difference between the completion time of the job C_{12} and the due date of the order d_1 because the due date is past the completion time of the order. In contrast, the completion time is later than the due date for order 2. Therefore, E_{22} is the difference between C_{22} and C_2. Moreover, if the completion time is past the due date, the tardiness will be larger than 0, as it is for order 4.

3.2 Problem properties and mixed-integer programming formulation

We consider the preemptive case first. Here, we are allowed to interrupt the processing of a job a at any time, continue with processing a job b or leave the machine idle, and then resume the processing of job a at any given time. By Proposition 2, we indicate that we can neglect the earliness for this problem configuration. To prove the proposition, we need the following lemma.

Lemma 1 *There always exists an optimal solution for the problem of minimizing the total tardiness in the dedicated machine environment without preemptions; i.e., $DPm \rightarrow 0|prmp|\sum T_i$ and $DPm \rightarrow 0||\sum T_i$ have a common optimal solution.*

Proof Assume two jobs of orders a and b on machine j which cannot start before time t_{start}. Since we consider the dedicated machine environment, we can observe the machines separately. If we include a preemption, i.e., interrupting the processing of job (a, j) after time t_x in order to process job (b, j), and resuming the processing of (a, j) afterward, we obtain the schedule (p) with the completion times

$$C_{aj}^{(p)} = t_{start} + p_{aj} + p_{bj}$$
$$C_{bj}^{(p)} = t_{start} + t_x + p_{bj}.$$

However, we can create the non-preemptive schedule (np), where (b, j) is scheduled directly before (a, j), and obtain the completion times

$$C_{aj}^{(np)} = t_{start} + p_{aj} + p_{bj}$$
$$C_{bj}^{(np)} = t_{start} + p_{bj}.$$

The completion times of the other jobs on the machine are not affected by this. It can be seen that in schedule (np), neither completion time is greater than the corresponding completion times in (p), i.e., $C_{aj}^{(np)} = C_{aj}^{(p)}$ and $C_{bj}^{(np)} \leq C_{bj}^{(p)}$. Consequently, the tardiness of the corresponding orders cannot be decreased by preemptive schedules since $T_i = \max\{0, \max_{1\leq j\leq m}\{C_{ij}\} - d_i\}$, and d_i is a given parameter. The proof for the case in which job (b, j) is also interrupted is analogous. □

Proposition 2 *The optimal objective function value of an instance for the problem $DPm \rightarrow 0|prmp|\sum\left(\sum E_{ij} + mT_i\right)$ is equivalent to the optimal objective function value of the same instance for the problem $DPm \rightarrow 0||\sum mT_i$.*

Proof If preemption is allowed, it is possible to stop the processing of any job a and leave an infinitesimally small portion of the processing time. The processing of a can be resumed at any given time without interfering with the completion time of the other jobs on the machine because the processing of the other jobs can be stopped for an infinitesimally small time to finish a and resume afterward. Consequently, the following strategy can be applied: First, the problem $DPm \rightarrow 0|prmp|\sum mT_i$, which is equivalent to $DPm \rightarrow 0||\sum mT_i$ (see Lemma 1), is solved. Subsequently, set $C_{ij} =$

$\max\{C_i, d_i\}$ for each job j of order i, by stopping the production of the job an infinitesimally small time before finishing it and placing the remaining portion of the processing time to $\max\{C_i, d_i\}$. Since $T_i = \max\{0, C_i - d_i\}$, this does not increase the tardiness of the orders, and the earliness of each job equals zero. □

Remark 1 The solution for $DPm \rightarrow 0||\sum mT_i$ is equivalent to $DPm \rightarrow 0||\sum T_i$, because m is just a constant, multiplied by the complete objective function $\sum T_i$.

Studies in the literature regarding solution methods for $DPm \rightarrow 0||\sum T_i$ are presented in Sect. 2. Clearly, $DPm \rightarrow 0|prmp|\sum\left(\sum E_{ij} + mT_i\right)$ is mainly interesting from a theoretical perspective because leaving an infinitesimally small portion of the processing time of a job is not possible in a real-world scenario. Furthermore, semi-finished goods whose processing is paused still have to be stored and hence also entail inventory costs, which would be counter to the goal of minimizing the earliness of the goods. Therefore, we limit the remainder of this paper to the problem configuration $DPm \rightarrow 0||\sum\left(\sum E_{ij} + mT_i\right)$.

This problem configuration has the following two properties which can be shown by counterexamples that are presented in Appendix A. They stand in contrast to problem properties of related problems, which were pointed out in Sect. 2.

Lemma 3 *There are instances of this problem configuration where it is not possible to schedule all orders in the same sequence on each machine to obtain an optimal solution.*

Lemma 4 *There are instances of this problem configuration where it is not possible to schedule the jobs without inserting idle times between the jobs to obtain an optimal solution, even if all orders have the same due date.*

Because of these properties, idle times must be considered, and each machine can have its own order sequence for job processing. The following MILP formulation considers these two properties.

$$\text{minimize: } \sum_{i=1}^{n}\left(\sum_{j=1}^{m}\left(E_{ij}\right) + m \cdot T_i\right) \tag{1}$$

subject to:

$$\sum_{h=1}^{n} x_{ijh} = 1 \,\forall\, i \in I;\; j \in J \tag{2}$$

$$\sum_{i=1}^{n} x_{ijh} = 1 \,\forall\, h \in I;\; j \in J \tag{3}$$

$$\sum_{i=1}^{n} z_{ij(h-1)} \leq \sum_{i=1}^{n}\left(z_{ijh} - x_{ijh} \cdot p_{ij}\right) \;\forall\, h \in I : h \neq 1;\; j \in J \tag{4}$$

$$0 \leq \sum_{i=1}^{n}\left(z_{ij1} - x_{ij1} \cdot p_{ij}\right) \;\forall\, j \in J \tag{5}$$

$$C_{ij} - M \cdot \left(1 - x_{ijh}\right) \leq z_{ijh} \,\forall\, i \in I;\; h \in I;\; j \in J \tag{6}$$

$$\sum_{h=1}^{n} z_{ijh} \leq C_{ij} \,\forall\, i \in I;\; j \in J \tag{7}$$

$$C_{ij} \leq C_i \,\forall\, i \in I;\; j \in J \tag{8}$$

$$C_i - d_i \leq T_i \,\forall\, i \in I \tag{9}$$

$$d_i - C_{ij} \leq E_{ij} \,\forall\, i \in I;\; j \in J \tag{10}$$

$$C_i - C_{ij} \leq E_{ij} \,\forall\, i \in I;\; j \in J \tag{11}$$

$$0 \leq E_{ij} \,\forall\, i \in I;\; j \in J \tag{12}$$

$$0 \leq T_i \,\forall\, i \in I \tag{13}$$

$$0 \leq z_{ijh} \,\forall\, i \in I;\; h \in I;\; j \in J \tag{14}$$

$$x_{ijh} \in \{0; 1\} \,\forall\, i \in I;\; h \in I;\; j \in J \tag{15}$$

$$z_{ijh} = \begin{cases} C_{ij} \text{ if job } j \text{ of order } i \text{ is at position } h \\ 0 \text{ otherwise.} \end{cases} \tag{16}$$

$$x_{ijh} = \begin{cases} 1 \text{ if job } j \text{ of order } i \text{ is at position } h \\ 0 \text{ otherwise.} \end{cases} \tag{17}$$

The variables and parameters C_{ij}, C_i, E_{ij}, T_i, p_{ij}, and d_i have already been introduced. In addition, we use the binary decision variable x_{ijh} that indicates whether the job j of order i is placed in position h on machine j, the continuous decision variable z_{ijh} that becomes C_{ij} when job j of order i is at position h or zero otherwise, and the parameter M which is a large number. Equation (1) defines the objective function. By Eqs. (2) and (3) it is defined that each order has exactly one position on each machine, and vice versa. On each machine, the job at position h cannot be processed until the job at position $(h-1)$ is finished or before time 0 when $h = 1$. This is defined by Eqs. (4) and (5). Equations (6) and (7) determine the relationship between C_{ij} and z_{ijh}, i.e., that $z_{ijh} = C_{ij}$ if $x_{ijh} = 1$, and $z_{ijh} = 0$ if $x_{ijh} = 0$. On the one hand, z_{ijh} must be equal to or larger than C_{ij} when job j of order i is scheduled at position h, since $(1 - x_{ijh}) = 0$ in this case. If job j of order i is not scheduled at position h, the left-hand side of the equation will be equal to or less than 0 because of M, and hence $z_{ijh} \geq 0$ in this case due to Eq. (14). On the other hand, the sum of z_{ijh} over all $h \in I$ has to be equal to or smaller than C_{ij} by Eq. (7), so if $z_{ijk} = C_{ij}$, then $z_{ijh} = 0 \,\forall\, h \in I : h \neq k$. Equations (8)–(11) define C_i, T_i, and E_{ij}; Eqs. (12)–(14), that E_{ij}, T_i, and z_{ijh} must be nonnegative; and Eq. (15), the binarity of x_{ijh}. An additional description for z_{ijh} and x_{ijh} is given by Eqs. (16) and (17).

The definition of M is important for the solvability of the model. The purpose of the large number is to guarantee that

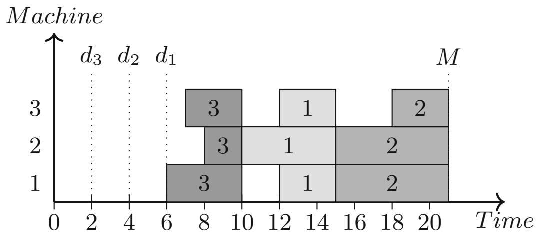

Fig. 2 Schedule with $\max_{1 \leq i \leq n}\{C_i\} = M$

Eq. (6) still holds when job j of order i is not at position h, i.e., when $x_{ijh} = 0$, and thus, $z_{ijh} = 0$. Rearranging Eq. (6) for this case, we obtain

$$C_{ij} \leq M \, \forall \, i \in I; \quad j \in J. \tag{18}$$

Consequently, M must be equal to or larger than each of the completion times of the jobs, i.e., the makespan of the schedule. In order to determine M, we present an upper bound for the completion times.

Lemma 5 *In an optimal solution, the completion time of each job does not exceed* $M = \sum_{i=1}^{n} \max_{1 \leq j \leq m}\{p_{ij}\} + \max_{1 \leq i \leq n}\{d_i\}$ *for each problem instance.*

Proof The proposed upper bound M can be achieved by scheduling all jobs past the last due date in such a way that the earliness of each job becomes zero; see Fig. 2. Consequently, it is to show that $\sum_{i=1}^{n} T_i$ cannot be decreased by increasing the completion time of any job past M. Switching jobs on a machine without changing the sum of idle times on the machine does not increase the makespan of the schedule, and hence, the sum of idle times must be increased in order to let the makespan exceed M, which does not reduce $\sum_{i=1}^{n} T_i$. $\square$

Because of Lemma 4, idle times must be considered to obtain an optimal solution. Lemma 6 gives an important property for inserting idle times before placed jobs. In this lemma, we consider a job-group g of consecutively scheduled jobs. In g, the jobs are not separated by idle times, i.e., $it_{kj} = 0 \, \forall \, (k, j) \in g : k \neq a, it_{aj} \geq 0$, with (a, j) being the first scheduled job in g. However, the last scheduled job of g is separated from the following job by an idle time larger zero. For the group g, n_g denotes the number of jobs in this group and v_g the number of jobs in g with an earliness of zero. Figure 3 shows an exemplary Gantt chart for a machine j on which several of the described job-groups can be found, e.g., $\{2, 1, 3\}$, $\{1, 3\}$, $\{3\}$, $\{5, 4\}$, and $\{4\}$.

Lemma 6 *Increasing the idle time before the first scheduled job in g decreases the earliness–tardiness of the schedule as long as* $v_g < \frac{n_g}{2m}$ *holds.*

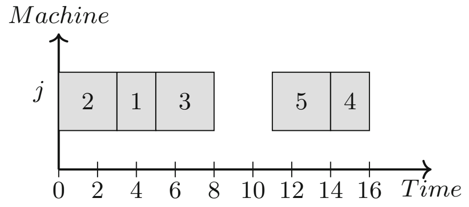

Fig. 3 Gantt chart showing exemplary job-groups

Proof Assume there is a job-group g on machine j, and the first scheduled job in g is (a, j). Increasing it_{aj} by $\delta \leq \min\{it_{fj}, E_{kj}\}$, where (f, j) is the job that is scheduled after the job-group and E_{kj} is the minimum earliness of the jobs in the job-group that is not zero, the earliness–tardiness decreases if

$$\delta(n_g - v_g) > \delta \cdot m \cdot v_g + \delta(m-1)v_g. \tag{19}$$

We denote the earliness and tardiness before the idle time insertion by E_{ij} and T_i, and after the idle time insertion by E'_{ij} and T'_i, respectively. The left-hand side of Eq. 19 gives the term that decreases the earliness–tardiness. The number of jobs with an earliness larger than zero before the idle time insertion is $n_g - v_g$. For each job (k, j) with $E_{kj} > 0$, the earliness is decreased by δ since $E'_{kj} = \max\{C_k, d_k\} - \left(C_{kj} + \delta\right) = E_{kj} - \delta$. The term $\max\{C_k, d_k\}$ remains the same for these jobs because $\delta \leq E_{kj} = \max\{d_k, C_k\} - C_{kj} \Leftrightarrow C_{kj} + \delta \leq \max\{d_k, C_k\}$, and consequently, the tardiness and further earlinesses of k do not change. For the other jobs in g, the earliness remains zero. The term that increases the earliness–tardiness is represented by the right-hand side of Eq. 19. For each job (l, j) with $E_{lj} = 0$, T_l increases by δ, since $C_{lj} = C_l \geq d_l$ (otherwise E_{lj} would not be zero), and hence, $T'_l = (C_l + \delta) - d_l = T_l + \delta$. Furthermore, the earliness of the others jobs of order l is increased by δ as well, because $E'_{lh} = (C_l + \delta) - C_{lh} = E_{lh} + \delta \, \forall \, h \in J : h \neq j$. Rearranging Eq. 19, we obtain

$$v_g < \frac{n_g}{2m}. \tag{20}$$

$\square$

Remark 2 As can be seen, Eq. 20 does not contain it_{aj} and δ. This means that the decision of whether an idle time should be increased before a job-group, i.e., the proposition of Lemma 6, does not directly depend on the value of δ. However, there is a limitation to the value of δ that is described in the corresponding proof. This limitation results from the requirement that v_g and n_g must remain the same for Eq. 19 to hold. Nevertheless, a change in v_g or n_g does not necessarily mean that a further increase in it_{aj} increases the objective value. It only indicates that Eq. 20 must be checked again for the new v_g and n_g.

Remark 3 As mentioned in the proof, for a job (i, j) with $E_{ij} \geq \delta$, it follows that $T_i = T_i^{'}$ and $E_{ik} = E_{ik}^{'} \forall k \in J : k \neq j$. Consequently, these values do not have to be recalculated when increasing an idle time by δ, which saves computation time for algorithms.

4 Heuristic approaches

We present six heuristics to approximately solve the problem $DPm \rightarrow 0|| \sum (\sum E_{ij} + mT_i)$. For the purpose of clarity, we first define some notation. Because of Lemma 4 and 3, the algorithms we develop consider idle times, and the orders are not necessarily processed in the same sequence on each machine. A sequence of jobs on machine j is denoted by $\pi_{ma,j}$, e.g., in Fig. 1 $\pi_{ma,2} = \{1, 2, 5, 4, 3\}$. Since it is clear which job of an order is considered on a given machine, the job is represented by its corresponding order. A set of these job sequences is denoted by π_{ma}. For the case in which all orders are processed in the same sequence on each machine, we introduce the order sequence π_{or}. Consequently, the sequences for the single machines can be derived from π_{or} by $\pi_{ma,j} := \pi_{or} \forall j \in J$. The set of idle times is designated as IT, i.e., $IT = \{it_{11}, it_{12}, ..., it_{1m}, it_{21}, ..., it_{nm}\}$. A complete schedule, and thus a solution to the problem, is described by its π_{ma} and IT. It is denoted by Π and, if it is the best schedule found, by Π^*. The objective function value of Π and Π^* is represented by $F(\Pi)$ and $F(\Pi^*)$, respectively. Furthermore, we use $F(\pi_{or})$ for the objective function value of a schedule in which all idle times are zero and each order is processed in the same sequence on each machine according to π_{or}.

Because of the good performance in Hoffmann et al. (2022), the six IGAs that we develop follow the concept of an IGA from that paper, which is called an *IGN*. For a general overview of the IGA, the reader is referred to Zhao et al. (2022). The heuristics developed herein have the typical IGA structure, with initialization, local search, destruction, construction, and acceptance. For further improvement, each IGA contains a different additional function which we label *Refinement*. The general procedure for the six IGAs is given in Algorithm 1. The determination of the four necessary parameters ds_{min}, ds_{max}, y, and z is described in Sect. 5.2.

A first solution is generated by the Initialization function; see Algorithm 2 for the pseudocode. Here, all orders are sorted in descending sequence of their due date. The first order of the resulting list is placed in an empty string π_{or}. Subsequently, the next order of the sorted list is inserted in every possible position of that string and is finally placed

Algorithm 1 General procedure for the IGAs

```
1:  procedure IGA($ds_{min}$, $ds_{max}$, $y$, $z$)
2:      $\pi_{or} \leftarrow Initialization()$
3:      $ds \leftarrow ds_{min}$
4:      $\pi_{or} \leftarrow LocalSearch(\pi_{or}, z, ds, ds_{min})$
5:      $\Pi \leftarrow Refinement(\pi_{or})$
6:      $\Pi^* \leftarrow \Pi$
7:      while time limit not reached do
8:          $\pi_{or:ds}, \pi_{or:n-ds} \leftarrow Destruction(\pi_{or}, ds)$
9:          $\pi_{or}^{'} \leftarrow Construction(\pi_{or:ds}, \pi_{or:n-ds})$
10:         $\pi_{or}^{'} \leftarrow LocalSearch(\pi_{or}^{'}, z, ds, ds_{min})$
11:         $\Pi^{'} \leftarrow Refinement(\pi_{or}^{'})$
12:         $\Pi^*, \Pi, \pi_{or}, ds \leftarrow Acceptance(\Pi^*, \Pi, \Pi^{'}, \pi_{or}, \pi_{or}^{'}, ds, ds_{min}, ds_{max}, y)$
13:     end while
14:     return $\Pi^*$, $F(\Pi^*)$
15: end procedure
```

Algorithm 2 Initialization function

```
1:  procedure INITIALIZATION( )
2:      Sort orders in descending order of $d_i$ in list
3:      $\pi_{or} \leftarrow \{\}$
4:      Insert the order at position 1 of list in $\pi_{or}$
5:      for $a \leftarrow 2$ to $n$ do
6:          for $b \leftarrow length(\pi_{or}) + 1$ to 1 do
7:              $\pi_{or,b} \leftarrow$ insert the order at position $a$ of list in $\pi_{or}$ at position $b$
8:          end for
9:          $\pi_{or} \leftarrow \pi_{or,b}$ with min. $F(\pi_{or,b})$
10:     end for
11:     return $\pi_{or}$
12: end procedure
```

in the position that leads to the minimum $F(\pi_{or})$. This is repeated until all orders are placed in π_{or}.

Next, the destruction size ds is set to ds_{min}. During the Local Search function, $\left\lfloor z \cdot \frac{ds}{ds_{min}} \right\rfloor$ different orders from π_{or} are chosen randomly and stored in a list without changing π_{or}. One after another, the orders from the list are taken out of π_{or} and are reinserted at the best possible position in π_{or} with respect to $F(\pi_{or})$. Algorithm 3 shows the corresponding pseudocode. Based on π_{or}, a schedule Π is generated by the Refinement function. This function varies between the proposed IGAs and will therefore be explained later. The solution obtained from this function is saved as Π^*.

The following loop starts with the Destruction function. As given in Algorithm 4, ds orders are taken out of π_{or} and are stored in $\pi_{or:ds}$. The rest of the orders are stored in the partial schedule $\pi_{or:n-ds}$ in the same sequence as they appear in π_{or}. In the following Construction function (see Algorithm 5 for the pseudocode), the orders in $\pi_{or:ds}$ are sorted in descending order by their due date. Subsequently, these orders are reinserted one after another at the best possible position in $\pi_{or:n-ds}$ with respect to $F(\pi_{or:n-ds})$. At the end of the Construction function, $\pi_{or}^{'}$ is assigned the final $\pi_{or:n-ds}$. Again,

Algorithm 3 Local Search function

```
1:  procedure LocalSearch($\pi_{or}, z, ds, ds_{min}$)
2:      $\pi_{or,copy} \leftarrow \pi_{or}$
3:      for $a \leftarrow 1$ to $\left\lfloor z \cdot \frac{ds}{ds_{min}} \right\rfloor$ do
4:          take a random order out of $\pi_{or,copy}$ and insert it in list
5:      end for
6:      for $a \leftarrow 1$ to $\left\lfloor z \cdot \frac{ds}{ds_{min}} \right\rfloor$ do
7:          $o \leftarrow$ order at position $a$ of list
8:          $\pi_{or,interim} \leftarrow \pi_{or} \setminus o$
9:          for $b \leftarrow n$ to 1 do
10:             $\pi_{or,b} \leftarrow$ insert $o$ in $\pi_{or,interim}$ at position $b$
11:         end for
12:         $\pi_{or} \leftarrow \pi_{or,b}$ with min. $F(\pi_{or,b})$
13:     end for
14:     return $\pi_{or}$
15: end procedure
```

Algorithm 4 Destruction function

```
1: procedure Destruction($\pi_{or}, ds$)
2:     $\pi_{or:n-ds}, \pi_{or:ds} \leftarrow \pi_{or}, \{\}$
3:     for $a \leftarrow 1$ to $ds$ do
4:         take a random order out of $\pi_{or:n-ds}$ and append it to $\pi_{or:ds}$
5:     end for
6:     return $\pi_{or:ds}, \pi_{or:n-ds}$
7: end procedure
```

Algorithm 5 Construction function

```
1:  procedure Construction($\pi_{or:ds}, \pi_{or:n-ds}$)
2:      Sort the orders of $\pi_{or:ds}$ in descending order of their $d_i$
3:      while $0 < length(\pi_{or:ds})$ do
4:          $o \leftarrow$ order at position 1 of $\pi_{or:ds}$
5:          for $a \leftarrow length(\pi_{or:n-ds}) + 1$ to 1 do
6:              $\pi_{or,a} \leftarrow$ insert $o$ in $\pi_{or:n-ds}$ at position $a$
7:          end for
8:          remove $o$ from $\pi_{or:ds}$
9:          $\pi_{or:n-ds} \leftarrow \pi_{or,a}$ with min. $F(\pi_{or,a})$
10:     end while
11:     $\pi'_{or} \leftarrow \pi_{or:n-ds}$
12:     return $\pi'_{or}$
13: end procedure
```

after applying the Local Search function, a feasible solution Π' is generated by the Refinement function based on π'_{or}.

The obtained Π' is evaluated by the Acceptance function. The pseudocode of this function is given in Algorithm 6. Π' is accepted as new Π if $F(\Pi') \leq F(\Pi)$. Consequently, π_{or} becomes π'_{or} in this case, and hence, a new order string is used in the next iteration. Furthermore, Π^* is assigned Π' if $F(\Pi') < F(\Pi^*)$. A solution Π' can also be accepted as Π if $F(\Pi') > F(\Pi)$, provided $q \leq e^{-y \frac{ds_{min}}{ds} \frac{F(\Pi')-F(\Pi)}{F(\Pi)}}$ holds, where q is a random number between 0 and 1, drawn in each iteration from a uniform distribution. In this case, π_{or} is assigned π'_{or}. In addition, for the case $F(\Pi') < F(\Pi)$, ds becomes ds_{min}; otherwise, ds is increased by 1 if $ds < \lfloor ds_{max} \rfloor$. The loop terminates when a given time limit is reached.

Algorithm 6 Acceptance function

```
1:  procedure Acceptance($\Pi^*, \Pi, \Pi', \pi_{or}, \pi'_{or}, ds, ds_{min}, ds_{max}, y$)
2:      if $F(\Pi') < F(\Pi)$ then
3:          $ds, \Pi, \pi_{or} \leftarrow ds_{min}, \Pi', \pi'_{or}$
4:          if $F(\Pi') < F(\Pi^*)$ then
5:              $\Pi^* \leftarrow \Pi'$
6:          end if
7:      else if $F(\Pi') = F(\Pi)$ then
8:          $\Pi, \pi_{or} \leftarrow \Pi', \pi'_{or}$
9:          if $ds < \lfloor ds_{max} \rfloor$ then
10:             $ds \leftarrow ds + 1$
11:         end if
12:     else
13:         if $q \leq e^{-y \frac{ds_{min}}{ds} \frac{F(\Pi')-F(\Pi)}{F(\Pi)}}$ then
14:             $\Pi, \pi_{or} \leftarrow \Pi', \pi'_{or}$
15:         end if
16:         if $ds < \lfloor ds_{max} \rfloor$ then
17:             $ds \leftarrow ds + 1$
18:         end if
19:     end if
20:     return $\Pi^*, \Pi, \pi_{or}, ds$
21: end procedure
```

In the following, the six Refinement functions are presented. An exemplary schedule before applying the Refinement function, based on $\pi_{or} = \{3, 1, 2\}$, is shown in Fig. 4. In general, the Refinement has three optional steps. First, in five of the six functions, offsets (i.e., idle times before the first jobs placed on the machines) are inserted. Subsequently, in four of the functions, swapping of jobs on single machines is checked, so it is possible that not all orders are processed in the same sequence on each machine afterward. In the last optional step, the idle times before each job are reviewed and increased if necessary. Algorithm 7 shows the structure of the general Refinement function. Note that not all steps are used in every proposed Refinement function.

Algorithm 7 General Refinement function

```
1:  procedure Refinement($\pi_{or}$)
2:      $IT \leftarrow \{0, 0, ..., 0\}$
3:      $\pi_{or}, IT \leftarrow Offset(\pi_{or}, IT)$
4:      $\pi_{ma} \leftarrow \pi_{or}$
5:      $\Pi \leftarrow \{\pi_{ma}, IT\}$
6:      $\Pi, \pi_{ma}, IT \leftarrow JobSwitching(\Pi, \pi_{ma}, IT)$
7:      $\pi_{ma}, IT \leftarrow IdleTimeInsertion(\pi_{ma}, IT)$
8:      $\Pi \leftarrow \{\pi_{ma}, IT\}$
9:      return $\Pi$
10: end procedure
```

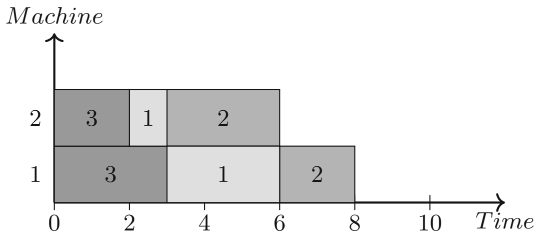

Fig. 4 Exemplary schedule before applying the Refinement function

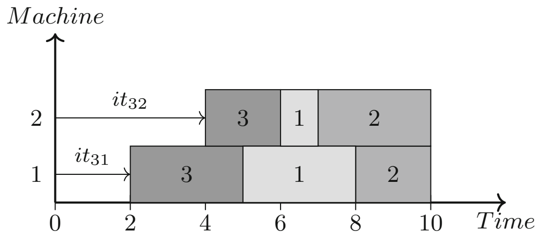

Fig. 5 Exemplary schedule after setting the offsets

Two different ways of defining the offsets were considered. The pseudocode for the corresponding functions is given in Algorithms 8 and 9. By using the Minimum Offset function, the minimum earliness of all jobs on a machine is identified. On each machine, the idle time before the first scheduled job is set to the corresponding minimum earliness on this machine, and the respective earliness and completion times are updated. Note that when applying this function, no tardiness is changed, and when an idle time is increased on a machine, the earliness on the other machines remains constant because of Remark 3.

Algorithm 8 Minimum Offset function

```
1: procedure MinimumOffset($\pi_{or}$, $IT$)
2:     $o \leftarrow$ order at position 1 of $\pi_{or}$
3:     for $a \leftarrow 1$ to $m$ do
4:         $E_a \leftarrow$ lowest $E_{ia}$ on machine $a$
5:         $it_{oa} \leftarrow E_a$
6:         Update $C_{ia}$ and $E_{ia}$ for all jobs on $a$
7:         Update $C_i$ for applicable orders
8:     end for
9:     return $\pi_{or}$, $IT$
10: end procedure
```

Algorithm 9 Calculated Offset function

```
1: procedure CalculatedOffset($\pi_{or}$, $IT$)
2:     for $a \leftarrow 1$ to $m$ do
3:         $E_a \leftarrow \lfloor \frac{n}{2 \cdot m} + 1 \rfloor$ lowest $E_{ia}$ on $a$
4:     end for
5:     Sort the machines in ascending order of $E_a$ in a list
6:     $o \leftarrow$ order at position 1 of $\pi_{or}$
7:     for $c \leftarrow 1$ to $m$ do
8:         $b \leftarrow$ machine in position $c$ of the list
9:         $E_b \leftarrow \lfloor \frac{n}{2 \cdot m} + 1 \rfloor$ lowest $E_{ib}$ on $b$
10:        $it_{ob} \leftarrow E_b$
11:        Update $C_{ib}$ and $E_{ib}$ for all jobs on $b$
12:        Update $C_i$ and $T_i$ if applicable
13:        Update $E_{ij}$ for all applicable jobs on each machine
14:    end for
15:    return $\pi_{or}$, $IT$
16: end procedure
```

The other function for setting an offset is called the Calculated Offset function. This function uses the property of Lemma 6. As mentioned in Remark 2, Eq. 20 only indicates whether a further increase in an idle time before a job-group decreases the objective function. However, the amount of the increase δ is not stated. Instead, δ is chosen in such a way that v_g or n_g is just changing. Subsequently, Eq. 20 must again be checked for a further increase in the idle time. To avoid the need to check Eq. 20 each time when v_g changes, the following strategy can be used as long as n_g remains the same: Increase the idle time before the first job of a job-group g, with the amount of the xth smallest earliness of the jobs in g, where $x = \lfloor \frac{n_g}{2 \cdot m} + 1 \rfloor$. It must be mentioned that if $\frac{n_g}{2 \cdot m} + 1$ results in an integer, this strategy increases the idle time even if $v_g = \frac{n_g}{2 \cdot m}$. In this case, the earliness–tardiness remains the same, because the decreasing part of Eq. 19 is equal to the increasing part of Eq. 19.

The Calculated Offset function uses this strategy in the following way. For each machine, the $\lfloor \frac{n}{2 \cdot m} + 1 \rfloor$ lowest earliness of the jobs on the machine is selected and saved as the factor for this machine. Next, the machines are sorted in ascending order by this factor in a list. The idle time before the first scheduled job on the first machine of the list is set to the corresponding factor of the machine. The corresponding earliness, tardiness, and completion times are updated afterward. One at a time, a machine is taken from the list and the corresponding factor of the machine is determined by selecting the $\lfloor \frac{n}{2 \cdot m} + 1 \rfloor$ lowest updated earliness of the machine. This factor is inserted as idle time before the first scheduled job on the machine and the relevant values of the schedule are updated again. This is repeated until all first idle times are set.

The result of both offset functions is a schedule where each order is processed in the same sequence on each machine, and only the idle times before the first scheduled jobs might be different from zero. Figure 5 illustrates this based on $\pi_{or} = \{3, 1, 2\}$ and $IT = \{0, 0, 0, 0, 2, 4\}$. In the next optional step, the Job Switching function is applied, whose pseudocode is given in Algorithm 10. Here, we check whether swapping adjacent jobs on single machines leads to a better solution. Consequently, the orders do not have to be processed in the same sequence on each machine afterward, and thus, π_{or} no longer describes the sequence of the jobs. Therefore, π_{or} is first transformed to a corresponding π_{ma}. The string $\pi_{ma,a}$

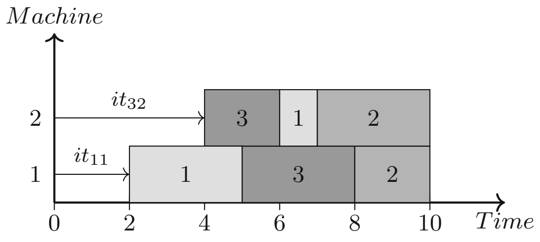

Fig. 6 Exemplary schedule after applying the Job Switching function

in π_{ma} represents the job sequence on machine a. Without prior sorting of the machines, one at a time, $\pi_{ma,a}$ is examined. Starting with the first scheduled job in $\pi_{ma,a}$, a job is swapped with its subsequent scheduled job and it is checked whether the determined schedule, with the previously determined offset, results in lower earliness–tardiness. Note that when the first scheduled job on a machine is swapped, the idle time before the new first scheduled job becomes the offset, and the idle time before the new second scheduled job becomes zero. If an improvement occurs, the jobs remain in this sequence; otherwise, the former sequence is retained. This is repeated until all but the last position of $\pi_{ma,a}$ have been tested by this swapping strategy. Therefore, $n-1$ swaps are checked for each $\pi_{ma,a}$. An exemplary resulting schedule $\Pi = \{\pi_{ma}, IT\} = \{\{\{1,3,2\},\{3,1,2\}\},\{2,0,0,0,0,4\}\}$ is depicted in Fig. 6.

Algorithm 10 Job Switching function

```
1: procedure JOBSWITCHING($\Pi, \pi_{ma}, IT$)
2:     for $a \leftarrow 1$ to $m$ do
3:         for $b \leftarrow 1$ to $n-1$ do
4:             $o_e \leftarrow$ job at position $b$ of $\pi_{ma,a}$
5:             $o_l \leftarrow$ job at position $(b+1)$ of $\pi_{ma,a}$
6:             $\pi_{test} \leftarrow \pi_{ma}$ where $o_e$ and $o_l$ are swapped in $\pi_{ma,a}$
7:             $IT_{test} \leftarrow IT$
8:             if $b = 1$ then
9:                 values of $it_{o_e}$ and $it_{o_l}$ exchange in $IT_{test}$
10:            end if
11:            $\Pi_{test} \leftarrow \{\pi_{test}, IT_{test}\}$
12:            if $F(\Pi_{test}) < F(\Pi)$ then
13:                $\Pi, \pi_{ma}, IT \leftarrow \Pi_{test}, \pi_{test}, IT_{test}$
14:            end if
15:        end for
16:    end for
17:    return $\Pi, \pi_{ma}, IT$
18: end procedure
```

The Idle Time Insertion function is performed as a last optional step and is presented in Algorithm 11. First, the machines are sorted in ascending order according to the sum of the earliness of the jobs on the respective machines. Subsequently, one after another, the machines are examined with their corresponding offsets and job sequences. Each time a machine q is examined, the earliness of the last scheduled job in $\pi_{ma,q}$ is determined. The idle time before the job is

Algorithm 11 Idle Time Insertion function

```
1: procedure IDLETIMEINSERTION($\pi_{ma}, IT$)
2:     for $a \leftarrow 1$ to $m$ do
3:         $E_a \leftarrow \sum_{i=1}^{n} E_{ia}$
4:     end for
5:     Sort machines in ascending order of $E_a$ in a list
6:     for $a \leftarrow 1$ to $m$ do
7:         $q \leftarrow$ machine in position $a$ of the list
8:         $l \leftarrow$ order of last scheduled job in $\pi_{ma,q}$
9:         $it_{lq} \leftarrow it_{lq} + E_{lq}$
10:        Update $C_{lq}$ and $E_{lq}$
11:        Update $C_l$ if applicable
12:        for $b \leftarrow n-1$ to 1 do
13:            $(o,q) \leftarrow$ job at position $b$ in $\pi_{ma,q}$
14:            $\kappa \leftarrow 1$
15:            while $\kappa = 1$ do
16:                $\kappa \leftarrow 0$
17:                Identify job-group $g$ with $(o,q)$ as first scheduled job
18:                $it_{fq} \leftarrow$ idle time before the job that follows $g$
19:                if $v_g < \frac{n_g}{2 \cdot m}$ then
20:                    $factor \leftarrow \lfloor \frac{n_g}{2 \cdot m} + 1 \rfloor$ lowest earliness in $g$
21:                    $\delta \leftarrow \min\{it_{fq}, factor\}$
22:                    $it_{oq} \leftarrow it_{oq} + \delta$
23:                    if $it_{fq} < factor$ then
24:                        $\kappa \leftarrow 1$
25:                    end if
26:                    $it_{fq} \leftarrow it_{fq} - \delta$
27:                    Update $C_{iq}$ and $E_{iq}$ in $g$
28:                    Update $C_i$ and $T_i$ for all applicable orders
29:                    Update $E_{ij}$ for all applicable jobs on each machine
30:                end if
31:            end while
32:        end for
33:    end for
34:    return $\pi_{ma}, IT$
35: end procedure
```

increased by the value of this earliness. Subsequently, the completion times of the job and, if applicable, the order are updated and the earliness is set to zero. The next steps are repeated for each of the other jobs in $\pi_{ma,q}$, starting with the second-to-last job, until the first scheduled job on q is examined. First, the job-group g of consecutively scheduled jobs, which is defined in the course of Lemma 6 and where the examined job (o,q) is scheduled first, is determined. Next, we follow a similar strategy as we did in the Calculated Offset function. If $v_g < \frac{n_g}{2 \cdot m}$, the $\lfloor \frac{n_g}{2 \cdot m} + 1 \rfloor$ lowest earliness of the jobs in g is selected. We denote the value of this earliness as $factor$ in the following. Subsequently, the minimum of $factor$ and the idle time separating g from the next scheduled group, it_{fq}, is determined. If the last scheduled job in g is also the last scheduled job in $\pi_{ma,q}$, it_{fq} is assumed to be infinity. The idle time before (o,q) is increased by this minimum and it_{fq} is decreased by this minimum. All affected earliness, tardiness, and completion times are updated. If it_{fq} was smaller than $factor$ before its adjustment, the strategy is repeated for the newly formed job-group until $it_{fq} \geq factor$ before the adjustment of it_{fq}. For the

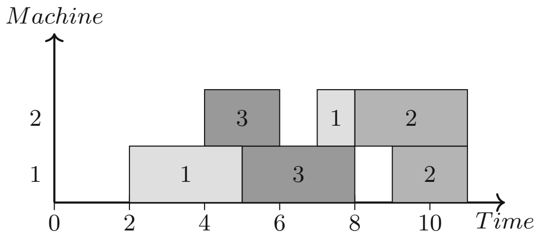

Fig. 7 Exemplary schedule after applying the Idle Time Insertion function

Table 2 Functions used for the Refinement function

Step	IGA1	IGA2	IGA3	IGA4	IGA5	IGA6
Offset	CO	MO	CO	No	MO	CO
JS	No	Yes	Yes	No	Yes	Yes
ITI	No	No	No	Yes	Yes	Yes

case in which $v_g \geq \frac{n_g}{2 \cdot m}$, the idle time before the examined job remains the same, and the preceding job in the sequence $\pi_{ma,q}$ is examined. Figure 7 illustrates an exemplary schedule after applying the Idle Time Insertion function, with $\Pi = \{\pi_{ma}, IT\} = \{\{\{1, 3, 2\}, \{3, 1, 2\}\}, \{2, 1, 1, 0, 0, 4\}\}$.

Using the functions described in Algorithms 8 - 11, six different Refinement functions, and hence six different IGAs, were created. In Table 2, the functions used for each IGA are given. Here, *MO* represents the Minimum Offset function, *CO* the Calculated Offset function, *JS* the Job Switching function, and *ITI* the Idle Time Insertion function. Note that in IGA4, no offset is determined, and the Refinement function begins directly with the Idle Time Insertion function.

5 Computational experiment

5.1 Experimental setting

In this section, we evaluate the IGAs that we developed in a computational experiment. Since the problem has not yet been studied in the literature, there are no comparable state-of-the-art algorithms. Consequently, the six IGAs are compared with each other and the MILP formulation from Sect. 3.2 solved by the Gurobi solver.

The test bed for the algorithm comparison is adopted from Framinan and Perez-Gonzalez (2018) and contains values for the processing times and due dates. The processing times of the jobs were randomly drawn from a uniform distribution $\mathcal{U}[1, 100]$, and the due dates from a uniform distribution $\mathcal{U}[P\left(1 - TF - \frac{RDD}{2}\right), P\left(1 - TF + \frac{RDD}{2}\right)]$, where $P = \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{p_{ij}}{m}$, and the range of due dates RDD and the tardiness factor TF are two given parameters for this instance. For easier understanding, we divide the original test bed into three parts and denote them as *Little*, *Medium*, and *Big*. The Little data set contains problem instances with $n \in \{10, 20\}$ and $m \in \{2, 5, 8\}$, the Medium data set problem instances with $n \in \{30, 40, 50\}$ and $m \in \{2, 5, 8\}$, and the Big data set problem instances with $n \in \{100, 150, 200, 300\}$ and $m \in \{5, 10\}$. For each problem size studied, $RDD \in \{0.2, 0.5, 0.8\}$ and $TF \in \{0.2, 0.5, 0.8\}$. In each data set, there are 20 different problem instances for each possible combination of n, m, RDD, and TF. By the combination $RDD \in \{0.5, 0.8\}$ and $TF \in \{0.8\}$, it is possible that negative due dates are drawn. In this case, a new due date was drawn during the instance generation until the due date was nonnegative.

The comparison criterion for the solution methods is the best earliness–tardiness value found after given run times. We set the time limit for the IGAs to $n\frac{m}{2}t\ sec$ and the corresponding time factor t to $t = 0.12$. Since the IGAs contain random elements, each instance was solved five times by each IGA during the parameter setting and the algorithm comparison. We set the time limit to 3600 *sec* for the solver to solve the MILP from Sect. 3.2 for the algorithm comparison. In contrast to the IGAs, each instance was solved only once by the solver.

For each run x, the relative percentage deviation (RPD) was calculated by

$$RPD_x = \frac{F(\Pi_x) - F(\Pi_{best})}{F(\Pi_{best})} \cdot 100\%, \tag{21}$$

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

All solution methods were implemented in Python. The calculations were run on an Intel Xeon CPU E5-2630 v2 2.60 GHz processor with 384 GB memory. To solve the MILP from Sect. 3.2 for the algorithm comparison, *Gurobi* 9 was used with a maximum of four threads in parallel. The graphics for the results of the computational experiment were created with IBM SPSS Statistics version 29 software.

5.2 Parameter setting

Each of the six IGAs uses four parameters: y, z, d_{min}, and d_{max}. As in Hoffmann et al. (2022), d_{min} and d_{max} were set to $d_{min} = 1$ and $d_{max} = \frac{n}{2}$. The other two parameters were determined experimentally. We limited the parameter setting to the problem sizes of the Little and the Medium data sets due to computational capacities. Since we used the original test bed later for the algorithm comparison, we generated 10 new instances per problem setting exclusively for the parameter setting.

Table 3 Average RPDs in % of the tested parameter values

	Value	IGA1	IGA2	IGA3	IGA4	IGA5	IGA6
y	100	15.22	**14.71**	10.30	7.67	8.36	**6.12**
	500	**14.70**	14.80	10.36	7.74	8.45	6.17
	1000	15.00	15.00	10.50	8.01	**7.37**	6.31
	5000	15.83	15.02	**9.50**	**7.22**	7.74	6.55
z	1	15.56	15.16	10.53	8.07	8.36	6.60
	1.25	15.30	14.99	10.22	7.81	8.11	6.38
	1.5	15.05	14.79	10.06	7.57	7.88	6.18
	1.75	**14.98**	**14.72**	**10.00**	7.43	7.78	**6.13**
	2	15.03	14.74	10.01	**7.42**	**7.76**	6.15

The lowest average RPD per algorithm and parameter are shown in bold

Table 4 Chosen parameter values for the IGAs

Parameter	IGA1	IGA2	IGA3	IGA4	IGA5	IGA6
y	500	100	5000	5000	1000	100
z	1.75	1.75	1.75	2	2	1.75

For similar reasons as in Hoffmann et al. (2022), z is limited to $[1, 2]$ and y must be greater than zero. After some pre-experiments, $z \in \{1, 1.25, 1.5, 1.75, 2\}$ and $y \in \{100, 500, 1000, 5000\}$ were considered for further investigation. Each IGA solved each generated problem instance five times with each parameter combination. Consequently, $6 \cdot 5 \cdot 3 \cdot 3 \cdot 3 \cdot 10 \cdot 5 \cdot 5 \cdot 4 = 810,000$ runs were performed for the parameter setting.

For each IGA, the average RPD was calculated separately for the different y-values and z-values across all runs. These averages are given in Table 3. The lowest average RPD per algorithm and parameter is shown in bold. For each algorithm and parameter, the parameter value is chosen that led to the lowest average RPD. The parameter selection is given in Table 4.

5.3 Algorithm evaluation

Despite the higher time limit of 3600 sec, the Gurobi solver could only confirm optimal solutions for instances with problem size $\{n, m\} = \{10, 2\}$. For this problem size, 120 of 180 problem instances were solved to optimality. The time for confirming the optimal solution by the Gurobi solver ranged from 7.42 sec to 3592.79 sec.

For the instances with problem size $\{n, m\} = \{20, 5\}$, the smallest gap determined by Gurobi was 17.51%, and for instances with problem size $\{n, m\} = \{20, 8\}$, it was 39.55%. Therefore, the solver solution was only considered for the instances in the Little data set. Consequently, there were 1080 runs with the MILP formulation, and for the IGAs, there were 32,400 runs for the Little data set, 48,600 runs for the Medium data set, and 43,200 runs for the Big data set in total.

The average RPDs of the solution methods for the different problem sizes are given in Table 5. To evaluate the overall performance of the IGAs, Fig. 8 shows the average RPDs of the IGAs across all problem instances of the Little, Medium, and Big data sets with 95% confidence intervals. Note that the calculation of the RPDs for the Little data set includes the solver solution.

The problem size has a notable influence on the performance of the solver solution. Regarding the number of orders n, this is indicated by Table 5. While the solver solution had a lower average RPD than the IGAs for $n = 10$, the reverse was found for the problem instances with $n = 20$. Furthermore, each of the IGAs had a lower average RPD than the solver solution for the complete Little data set.

As depicted in Fig. 8, IGA6 had the best performance among all IGAs across all examined problem instances. However, as for the solver, the performance of the IGAs depended on the problem size. Table 5 shows that IGA6 had the best performance for the Little and the Medium data sets but was outperformed by IGA3 and IGA4 for the Big data set. To be more precise, IGA6 outperformed the other IGAs for $n \le 100$, while the best-performing algorithm for $n = 150$ was IGA4, and for $n \ge 200$ it was IGA3. Furthermore, the average RPD of IGA1 decreased for a higher number of orders.

We assume by this that both the Idle Time Insertion function and the Job Switching function occupy too much computation time to explore the solution space properly in the given time limit for larger values of n. We furthermore conclude that the general structure of the IGA itself works properly for the studied problem configuration of the COSP since it becomes more prominent as n increases.

In Figs. 9 and 10, the average RPDs with 95% confidence interval for the solution methods per number of machines are illustrated for the Little data set and the Medium data set, respectively. The illustration for the Big data set is given in the appendix (Fig. 15).

As shown in Fig. 9, the solver solution had a lower average RPD than the IGAs for the problem instances from the Little data set with $m = 2$. However, for the problem instances of the Little data set with $m \ge 5$, each IGA outperformed the solver solution.

Figure 10 shows that the performance of the algorithms with the Idle Time Insertion function increases with the number of machines. This can be seen by the expanding gaps between IGA2 and IGA5 and between IGA3 and IGA6, since the structures of IGA2 and IGA5 and the those of IGA3 and IGA6 differ only in the additional Idle Time Insertion function. Furthermore, Fig. 10 shows that the performance of algorithms with a lower offset setting function increases with higher m because the gap between algorithms with the

Table 5 Average RPDs in % of the solution methods for different problem sizes

n	m	MILP	IGA1	IGA2	IGA3	IGA4	IGA5	IGA6
10	2	0.14	23.50	25.13	16.78	15.67	15.00	10.70
	5	5.61	25.49	21.31	19.34	11.63	10.49	9.50
	8	14.04	24.48	18.42	18.46	6.28	6.70	6.37
	Avg.	6.60	24.49	21.62	18.19	11.19	10.73	8.86
20	2	3.37	28.65	33.22	21.92	24.44	24.84	18.65
	5	42.21	15.42	14.16	10.16	7.02	7.21	4.46
	8	72.12	15.02	12.14	9.93	4.13	5.21	3.82
	Avg.	39.23	19.70	19.84	14.00	11.86	12.42	8.98
Avg. *Little*		22.91	22.09	20.73	16.10	11.53	11.58	8.92
30	2	–	9.92	11.63	6.13	6.59	6.69	4.03
	5	–	11.85	11.26	7.50	5.50	6.23	3.46
	8	–	12.81	10.44	8.52	4.07	5.22	3.76
	Avg.	–	11.53	11.11	7.38	5.39	6.05	3.75
40	2	–	9.35	11.19	6.27	7.23	7.13	4.20
	5	–	10.38	10.56	6.90	5.23	6.36	3.69
	8	–	11.09	10.20	7.55	4.43	5.96	3.74
	Avg.	–	10.27	10.65	6.91	5.63	6.48	3.88
50	2	–	9.11	10.09	6.08	7.27	6.84	4.37
	5	–	9.34	9.67	6.45	5.40	6.19	3.44
	8	–	9.75	9.82	6.94	4.23	6.21	3.88
	Avg.	–	9.40	9.86	6.49	5.63	6.41	3.90
Avg. *Med.*		–	10.40	10.54	6.93	5.55	6.31	3.84
100	5	–	6.92	8.35	4.70	4.22	5.63	3.83
	10	–	6.47	7.46	4.44	3.50	5.09	3.46
	Avg.	–	6.69	7.90	4.57	3.86	5.36	3.65
150	5	–	6.47	8.04	5.19	5.66	6.31	5.51
	10	–	5.58	7.16	4.05	3.44	5.41	4.37
	Avg.	–	6.03	7.60	4.62	4.55	5.86	4.94
200	5	–	5.71	7.64	4.50	5.66	6.44	5.83
	10	–	5.38	7.44	3.91	3.58	5.75	4.51
	Avg.	–	5.55	7.54	4.20	4.62	6.10	5.17
300	5	–	4.94	5.86	4.39	5.37	5.55	5.72
	10	–	4.59	6.32	4.35	4.92	6.21	6.10
	Avg.	–	4.76	6.09	4.37	5.14	5.88	5.91
Avg. *Big*		–	5.76	7.28	4.44	4.54	5.80	4.92
Total avg.		–	11.84	12.07	8.45	6.76	7.51	5.54

Minimum Offset function, i.e., IGA2 and IGA5, and the corresponding algorithms with the Calculated Offset function, i.e., IGA3 and IGA6, narrows for larger values of m. Both statements are supported by the observation that IGA4, which has no offset setting function but the Idle Time Insertion function, performs better for a larger number of machines, and for $m = 8$ in the Medium data set it has nearly the same average RPD as IGA6.

Given these observations, we conclude that a precise idle time insertion for each single job becomes more important for a higher number of jobs per order. It remains open whether

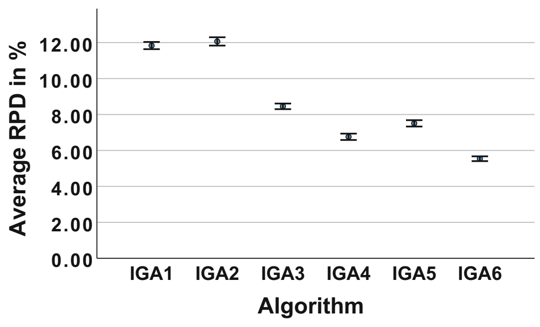

Fig. 8 Average RPDs of the IGAs for all problem instances

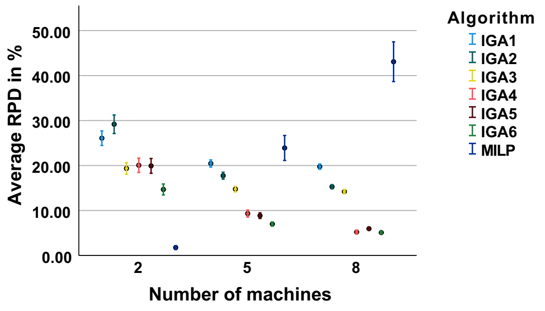

Fig. 9 Average RPDs per number of machines for the Little data set

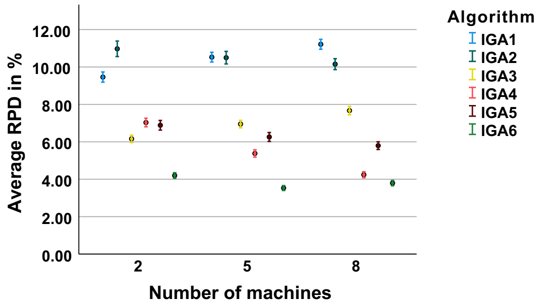

Fig. 10 Average RPDs per number of machines for the Medium data set

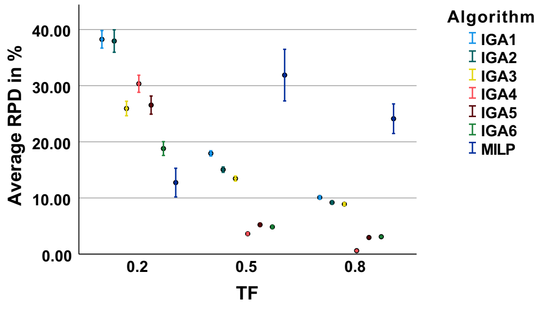

Fig. 11 Average RPDs per TF value for the Little data set

IGA4 outperforms the other IGAs for $m > 8$ when the other problem-defining parameters, i.e. n, TF, and RDD, remain the same as in the Medium data set.

Figures 11 and 12 present the average RPDs with 95% confidence intervals for the solution methods in terms of different values of the tardiness factor TF for the Little data set and the Medium data set, respectively. For the sake of completeness, the diagram for the Big data set is provided in the appendix (Fig. 16). In addition, the average RPDs of the solution methods for the different due date-generating parameters TF and RDD are presented in Tables 8, 9, and 10 for each data set.

Comparing the solver solution with the IGAs, Fig. 11 and Table 8 show that the average RPDs of the IGAs are higher for lower TF values. However, each of the IGAs outperformed

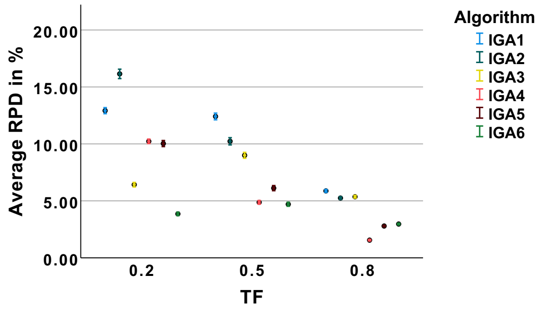

Fig. 12 Average RPDs per TF value for the Medium data set

the solver solution for the Little data set when the TF value was 0.5 or greater.

Figure 12 shows that IGA3 performs better than IGA1 and IGA2, and IGA6 performs better than IGA5 for $TF = 0.2$ in the Medium data set. However, the gap between IGA1 and IGA3, the gap between IGA2 and IGA3, and the gap between IGA5 and IGA6 narrow for higher TF values, and for $TF = 0.8$, IGA2 even performs better than IGA3, and IGA5 performs better than IGA6. Furthermore, the average RPD of IGA4 is 6.37 percentage points higher than the average RPD of IGA6 for $TF = 0.2$ in the Medium data set, while it is 1.42 percentage points lower for $TF = 0.8$; see Table 9 for the average RPD values. For $TF = 0.8$, IGA4 is also the best-performing IGA in the Little, Medium, and Big data sets.

On the basis of these findings, we conclude that a high offset and the Job Switching function improve the algorithm quality for low TF values, while omitting offset setting and the Job Switching function is advisable for high TF values. We assume that the reason for the good performance of lower offsets for higher TF values is that higher TF values on average lead to smaller due dates in the data set. Consequently, the due dates of the orders are reached with lower offsets, and hence, higher offsets lead to higher tardiness of the orders. Instead, a more precise idle time setting with the Idle Time Insertion function is preferable for data sets with a high TF value.

The average RPDs with 95% confidence intervals for the solution methods per RDD value are illustrated in Fig. 13 for the Little data set, in Fig. 14 for the Medium data set, and in Fig. 17 for the Big data set.

Comparing the solution methods for the Little data set regarding the RDD values, Fig. 13 shows that for each solution method, the average RPD increases for higher RDD values. We conclude from this that for a higher RDD value, the solution quality of the solution methods becomes more volatile. This can also be seen by the larger confidence intervals for greater RDD values in Fig. 13.

As in the Little data set, the average RPD increases for higher RDD values for each IGA in the Medium data set.

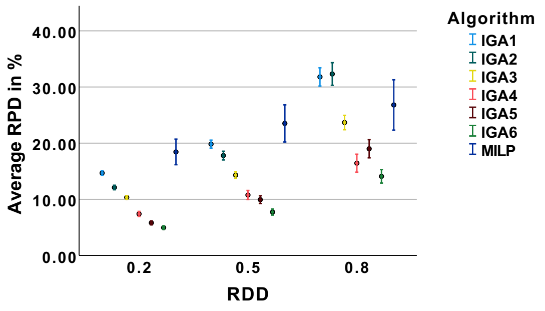

Fig. 13 Average RPDs per RDD value for the Little data set

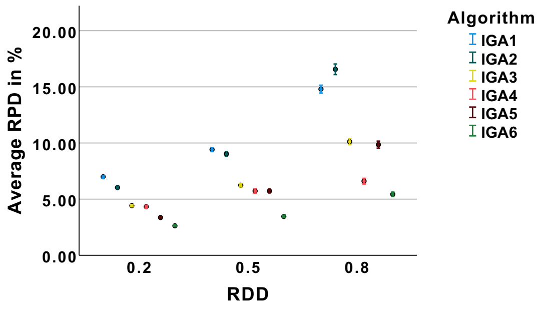

Fig. 14 Average RPDs per RDD value for the Medium data set

Furthermore, the gap between IGA2 and IGA3 and the gap between IGA5 and IGA6 increase for higher RDD values. This might lead to the assumption that higher offsets are advisable for data sets with a high RDD value. However, the gap between IGA4, which uses no offset setting function, and IGA6 remains nearly the same for increasing RDD values, or even narrows. It is left for future research to examine this finding in greater detail. According to the results of the experiment, we limit ourselves to the conclusion that the Calculated Offset function should be applied before using the Job Switching function, especially if the due dates in the data set have a high range.

6 Conclusion

In this paper, we studied the earliness–tardiness minimization of the COSP in the dedicated machine environment. Properties of this problem configuration were derived, and based on these, a MILP was formulated and six IGAs were developed to solve the problem. The IGAs differ in their Refinement function, which may consist of an offset setting function, the Job Switching function, and the Idle Time Insertion function.

In a computational experiment, we compared the developed heuristics with each other and the MILP solution from the Gurobi solver. For the smallest problem size studied, with two machines and 10 orders, the solver was able to find and confirm optimal solutions for 66.67% of the instances.

Although solving the MILP with the solver provided good solutions in reasonable time for a small number of orders, each of the IGAs outperformed the solver solution for a number of orders of 20. Furthermore, solving the MILP with Gurobi led to better solutions when the tardiness factor of the data was low, i.e., the due dates were high. However, the IGAs outperformed the solver for a tardiness factor of 0.5 or higher.

Comparing the IGAs with each other, IGA6, which uses the Idle Time Insertion function, the Calculated Offset function, and the Job Switching function, had the best overall performance. However, for a high tardiness factor value, IGA4, which uses only the Idle Time Insertion function, is preferable to the other IGAs, while for a large number of orders, IGA3 performed better, by using the Calculated Offset function and the Job Switching function.

Further studies may investigate Refinement functions that are more suitable for the case of low tardiness factor values. Furthermore, other metaheuristic approaches or problem-specific heuristics could be developed for this problem configuration of the COSP and tested against the proposed IGAs. In addition, since the earliness–tardiness is not broadly studied for the COSP, future research could consider the earliness–tardiness in other machine environments, such as the identical machine environment.

Appendix A Proofs

A.1 Proof of Lemma 3

Lemma 3 can be shown by counterexamples. One is presented in detail below.

Proof If Lemma 3 is not true, there exists an optimal solution for each problem instance where the orders are processed in the same sequence on each machine. However, for the example given in Table 6, this is not true.

The given instance can be solved by the Gurobi Optimizer for two different formulations. The first formulation is presented in Sect. 3.2 and allows one to schedule orders in different sequences on the machines. The other formulation can be found below. By this formulation, all orders must be in the same sequence on each machine. The decision variable x_{ih} indicates whether the order i is placed in position h on all machines, while C_{hj} is the completion time of the job j from the order that is placed in position h. The tardiness of the

Table 6 Problem instance of the counterexample for Lemma 3

p_{ij}	p_{i1}	p_{i2}	d_i
O_1	2	1	7
O_2	1	1	8
O_3	1	6	9

order in position h is denoted by T_h and the earliness of that order on machine j by E_{hj}. In Eq. A1, the objective function is defined. In contrast to the formulation from Sect. 3.2, Eqs. A2 and A3 assign one position for all machines to each order, and vice versa. Also, Eqs. A4 and A5 were adjusted to the new decision variables, in which the job of an order in position $h-1$ must be finished before the next job can be processed on that machine, and the jobs of the order in the first position must be processed after time 0. By Eqs. A6 - A10, the tardiness and the earliness are defined, and Eq. A11 guarantees the binarity of x_{ih}.

$$\text{minimize: } \sum_{h=1}^{n}\left(\sum_{j=1}^{m}\left(E_{hj}\right)+m\cdot T_h\right) \tag{A1}$$

subject to:

$$\sum_{h=1}^{n} x_{ih}=1\,\forall\, i\in I \tag{A2}$$

$$\sum_{i=1}^{n} x_{ih}=1\,\forall\, h\in I \tag{A3}$$

$$C_{(h-1)j}\le C_{hj}-\sum_{i=1}^{n} x_{ih}\cdot p_{ij}\,\forall\, h\in I: h\neq 1;\; j\in J \tag{A4}$$

$$0\le C_{1j}-\sum_{i=1}^{n} x_{i1}\cdot p_{ij}\,\forall\, j\in J \tag{A5}$$

$$C_{hj}-\sum_{i=1}^{n} x_{ih}\cdot d_i\le T_h\,\forall\, h\in I;\; j\in J \tag{A6}$$

$$\sum_{i=1}^{n}\left(x_{ih}\cdot d_i\right)-C_{hj}\le E_{hj}\,\forall\, h\in I;\; j\in J \tag{A7}$$

$$C_{hk}-C_{hj}\le E_{hj}\,\forall\, h\in I;\; j\in J;\; k\in J \tag{A8}$$

$$0\le T_h\,\forall\, h\in I \tag{A9}$$

$$0\le E_{hj}\,\forall\, h\in I;\; j\in J \tag{A10}$$

$$x_{ih}\in\{0;1\}\,\forall\, i\in I;\; h\in I \tag{A11}$$

$$x_{ih}=\begin{cases}1 \text{ if order } i \text{ is at position } h\\ 0 \text{ otherwise.}\end{cases} \tag{A12}$$

The Gurobi Optimizer finds an optimal solution for both formulations. For the formulation from Sect. 3.2, the objective function value is 3 and the schedule is $\Pi_1 = \{\pi_{ma}, IT\} = \{\{\{1,2,3\},\{3,1,2\}\},\{5,0,0,0,0,0\}\}$, and for the formulation from this section, the objective function value is 7 and the schedule is $\Pi_2 = \{\pi_{ma}, IT\} = \{\{\{3,1,2\},\{3,1,2\}\},\{0,0,0,0,4,0\}\}$.

Consequently, there are instances in which the optimal solution is not a schedule in which all orders are processed in the same sequence on each machine, and Lemma 3 holds. □

Table 7 Problem instance of the counterexample for Lemma 4

p_{ij}	p_{i1}	p_{i2}	d_i
O_1	8	7	10
O_2	6	2	10

The MILP formulation in this section can be used for solving the problem configuration when it is required to process the orders in the same sequence on each machine, which might be necessary in real-world production environments.

A.2 Proof of Lemma 4

Lemma 4 can also be shown by counterexamples. One is presented in detail below.

Proof If Lemma 4 is not true, there exists an optimal solution for each problem instance with common due dates where no idle times between two jobs are inserted. However, for the example given in Table 7, this is not true.

The given instance can be solved by the Gurobi Optimizer for two different formulations. The first formulation is presented in Sect. 3.2 and allows one to insert idle time between two jobs. By the other formulation, no idle time can be inserted between two jobs on a machine. However, it is still possible to insert an idle time before the first scheduled job on each machine. The formulation differs from the formulation from Sect. 3.2 by adding the following constraint:

$$\sum_{i=1}^{n} z_{ij(h-1)}\ge\sum_{i=1}^{n}\left(z_{ijh}-x_{ijh}\cdot p_{ij}\right)\quad \forall\, h\in I: h\neq 1;\; j\in J. \tag{13}$$

The Gurobi Optimizer finds an optimal solution for both formulations. For the formulation from Sect. 3.2, the objective function value is 10 and the schedule is $\Pi_1 = \{\pi_{ma}, IT\} = \{\{\{1,2\},\{1,2\}\},\{0,3,0,2\}\}$, and for the formulation from this section, the objective function value is 12 and the schedule is $\Pi_2 = \{\pi_{ma}, IT\} = \{\{\{1,2\},\{1,2\}\},\{0,3,0,0\}\}$.

Consequently, there are instances in which it is not possible to schedule the jobs without inserting idle times between two jobs to obtain the optimal solution, even if all orders have the same due date, and hence, Lemma 4 holds. □

The MILP formulation in this section can still be used for solving the problem configuration when machine idle times are not allowed, which might be the case in real-world production environments.

Appendix B Tables and graphs

See Figs. 15, 16, and 17 and Tables 8, 9, and 10.

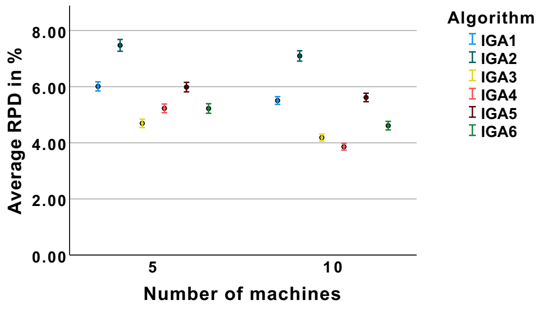

Fig. 15 Average RPDs per number of machines for the Big data set

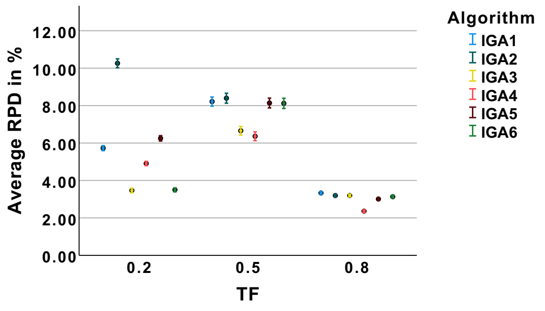

Fig. 16 Average RPDs per TF value for the Big data set

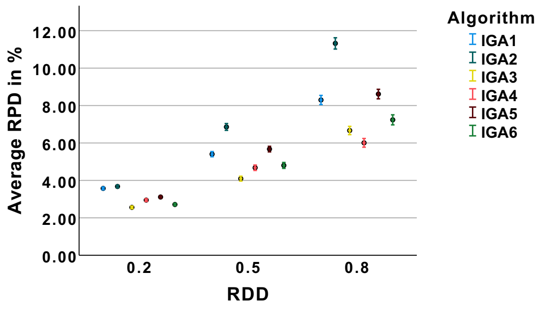

Fig. 17 Average RPDs per RDD value for the Big data set

Table 8 Average RPDs in % for different TF and RDD values for the Little data set

TF	RDD	MILP	IGA1	IGA2	IGA3	IGA4	IGA5	IGA6
0.2	0.2	11.74	20.81	16.95	13.04	17.32	11.17	8.68
	0.5	15.36	32.74	30.36	21.62	28.03	22.04	15.75
	0.8	11.13	61.20	66.54	43.16	45.68	46.41	31.97
		12.74	38.25	37.95	25.94	30.34	26.54	18.80
0.5	0.2	23.42	14.48	11.37	10.23	4.26	3.80	3.55
	0.5	32.07	17.03	14.08	12.65	3.83	4.90	4.42
	0.8	40.15	22.31	19.69	17.50	2.79	6.98	6.57
		31.88	17.94	15.05	13.46	3.62	5.23	4.85
0.8	0.2	20.15	8.73	7.97	7.72	0.60	2.42	2.61
	0.5	23.10	9.67	8.91	8.63	0.43	2.85	2.98
	0.8	29.10	11.87	10.70	10.35	0.81	3.61	3.71
		24.12	10.09	9.19	8.90	0.61	2.96	3.10
		22.91	22.09	20.73	16.10	11.53	11.58	8.92

Table 9 Average RPDs in % for different TF and RDD values for the Medium data set

TF	RDD	IGA1	IGA2	IGA3	IGA4	IGA5	IGA6
0.2	0.2	7.61	7.23	3.28	7.10	4.28	2.06
	0.5	11.55	13.57	5.60	10.46	9.35	3.60
	0.8	19.60	27.64	10.39	13.12	16.46	5.91
		12.92	16.15	6.42	10.23	10.03	3.86
0.5	0.2	8.43	6.52	5.64	4.44	3.49	3.34
	0.5	11.28	8.62	8.08	5.41	5.27	4.03
	0.8	17.52	15.54	13.27	4.77	9.60	6.73
		12.41	10.23	9.00	4.87	6.12	4.70
0.8	0.2	4.93	4.35	4.34	1.43	2.31	2.49
	0.5	5.42	4.87	5.04	1.32	2.55	2.74
	0.8	7.27	6.52	6.70	1.91	3.51	3.67
		5.87	5.25	5.36	1.55	2.79	2.97
		10.40	10.54	6.93	5.55	6.31	3.84

Table 10 Average RPDs in % for different TF and RDD values for the Big data set

TF	RDD	IGA1	IGA2	IGA3	IGA4	IGA5	IGA6
0.2	0.2	3.46	4.48	1.64	3.17	3.01	2.10
	0.5	5.69	10.22	3.19	5.47	6.84	3.93
	0.8	8.03	16.09	5.57	6.09	8.91	4.46
		5.73	10.26	3.47	4.91	6.25	3.50
0.5	0.2	4.30	3.83	3.33	3.57	3.74	3.52
	0.5	7.50	7.54	6.12	6.42	7.34	7.63
	0.8	12.84	13.82	10.52	9.10	13.34	13.21
		8.22	8.40	6.66	6.36	8.14	8.12
0.8	0.2	2.96	2.72	2.70	2.10	2.59	2.51
	0.5	3.01	2.81	2.97	2.14	2.84	2.85
	0.8	4.02	4.05	3.92	2.84	3.60	4.03
		3.33	3.20	3.20	2.36	3.01	3.13
		5.76	7.28	4.44	4.54	5.80	4.92

Acknowledgements We would like to thank Jose M. Framinan and Paz Perez-Gonzalez for providing the problem instances for the computational experiment.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

References

Antonioli, M. P., Rodrigues, C. D., & Prata, B. A. (2022). Minimizing total tardiness for the order scheduling problem with sequence-dependent setup times using hybrid matheuristics. *International Journal of Industrial Engineering Computations, 13*(2), 223–236. https://doi.org/10.5267/j.ijiec.2021.11.002

Bank, J., & Werner, F. (2001). Heuristic algorithms for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties. *Mathematical and Computer Modelling, 33*(4–5), 363–383. https://doi.org/10.1016/S0895-7177(00)00250-8

Braga-Santos, S. A., Barroso, G. C., & Prata, B. A. (2022). A size-reduction algorithm for the order scheduling problem with total tardiness minimization. *Journal of Project Management, 7*(3), 167–176. https://doi.org/10.5267/j.jpm.2022.1.001

Dauod, H., Li, D., Yoon, S. W., et al. (2018). Multi-objective optimization of the order scheduling problem in mail-order pharmacy automation systems. *The International Journal of Advanced Manufacturing Technology, 99*(1–4), 73–83. https://doi.org/10.1007/s00170-016-9123-1

de Abreu, L. R., de Athayde, P. B., Gomes, A. C., et al. (2022). A novel BRKGA for the customer order scheduling with missing operations to minimize total tardiness. *Swarm and Evolutionary Computation, 75*(101), 149. https://doi.org/10.1016/j.swevo.2022.101149

Framinan, J. M., & Perez-Gonzalez, P. (2017). New approximate algorithms for the customer order scheduling problem with total completion time objective. *Computers & Operations Research, 78*, 181–192. https://doi.org/10.1016/j.cor.2016.09.010

Framinan, J. M., & Perez-Gonzalez, P. (2018). Order scheduling with tardiness objective: Improved approximate solutions. *European Journal of Operational Research, 266*(3), 840–850. https://doi.org/10.1016/j.ejor.2017.10.064

Framinan, J. M., Perez-Gonzalez, P., & Fernandez-Viagas, V. (2019). Deterministic assembly scheduling problems: A review and classification of concurrent-type scheduling models and solution procedures. *European Journal of Operational Research, 273*(2), 401–417. https://doi.org/10.1016/j.ejor.2018.04.033

Gerstl, E., & Mosheiov, G. (2012). Scheduling job classes on uniform machines. *Computers & Operations Research, 39*(9), 1927–1932. https://doi.org/10.1016/j.cor.2011.08.004

Hazır, O., Günalay, Y., & Erel, E. (2008). Customer order scheduling problem: A comparative metaheuristics study. *The International Journal of Advanced Manufacturing Technology, 37*(5–6), 589–598. https://doi.org/10.1007/s00170-007-0998-8

Hoffmann, J., Neufeld, J. S., & Buscher, U. (2022). Iterated greedy algorithms for customer order scheduling with dedicated machines. *IFAC-PapersOnLine, 55*(10), 1594–1599. https://doi.org/10.1016/j.ifacol.2022.09.618

Kedad-Sidhoum, S., & Sourd, F. (2010). Fast neighborhood search for the single machine earliness-tardiness scheduling problem. *Computers & Operations Research, 37*(8), 1464–1471. https://doi.org/10.1016/j.cor.2009.11.002

Kramer, A., & Subramanian, A. (2019). A unified heuristic and an annotated bibliography for a large class of earliness-tardiness scheduling problems. *Journal of Scheduling, 22*(1), 21–57. https://doi.org/10.1007/s10951-017-0549-6

Kung, J. Y., Duan, J., Xu, J., et al. (2018). Metaheuristics for order scheduling problem with unequal ready times. *Discrete Dynamics in Nature and Society, 2018*, 1–13. https://doi.org/10.1155/2018/4657368

Lee, I. S. (2013). Minimizing total tardiness for the order scheduling problem. *International Journal of Production Economics, 144*(1), 128–134. https://doi.org/10.1016/j.ijpe.2013.01.025

Leung, J. Y. T., Li, H., & Pinedo, M. (2005). Order scheduling in an environment with dedicated resources in parallel. *Journal of Scheduling, 8*(5), 355–386. https://doi.org/10.1007/s10951-005-2860-x

Leung, J. Y. T., Li, H., Pinedo, M., et al. (2007). Minimizing total weighted completion time when scheduling orders in a flexible environment with uniform machines. *Information Processing Letters, 103*(3), 119–129. https://doi.org/10.1016/j.ipl.2007.03.002

Leung, J. Y. T., Lee, C., Ng, C., et al. (2008). Preemptive multiprocessor order scheduling to minimize total weighted flowtime. *European Journal of Operational Research, 190*(1), 40–51. https://doi.org/10.1016/j.ejor.2007.05.052

Leung, J. Y. T., Li, H., & Pinedo, M. (2008). Scheduling orders on either dedicated or flexible machines in parallel to minimize total weighted completion time. *Annals of Operations Research, 159*(1), 107–123. https://doi.org/10.1007/s10479-007-0270-5

Li, D., & Yoon, S. W. (2015). A novel fill-time window minimisation problem and adaptive parallel tabu search algorithm in mail-order pharmacy automation system. *International Journal of Production Research, 53*(14), 4189–4205. https://doi.org/10.1080/00207543.2014.985392

Li, D., Chen, K., Da, T., et al. (2018). Medication planogram design to minimize collation delays and makespan in parallel pharmaceutical automatic dispensing machines. *The International Journal of Advanced Manufacturing Technology, 99*(9–12), 2171–2180. https://doi.org/10.1007/s00170-018-2222-4

Li, L. Y., Xu, J. Y., Cheng, S. R., et al. (2022). A genetic hyper-heuristic for an order scheduling problem with two scenario-dependent parameters in a parallel-machine environment. *Mathematics, 10*(21), 4146. https://doi.org/10.3390/math10214146

Li, L. Y., Lin, W. C., Bai, D., et al. (2023). Composite heuristics and water wave optimality algorithms for tri-criteria multiple job classes and customer order scheduling on a single machine. *International Journal of Industrial Engineering Computations, 14*(2), 265–274. https://doi.org/10.5267/j.ijiec.2023.2.002

Liu, C. H. (2009). Lot streaming for customer order scheduling problem in job shop environments. *International Journal of Computer Integrated Manufacturing, 22*(9), 890–907. https://doi.org/10.1080/09511920902866104

Polyakovskiy, S., & M'Hallah, R. (2014). A multi-agent system for the weighted earliness tardiness parallel machine problem. *Computers & Operations Research, 44*, 115–136. https://doi.org/10.1016/j.cor.2013.10.013

Roemer, T. A. (2006). A note on the complexity of the concurrent open shop problem. *Journal of Scheduling, 9*(4), 389–396. https://doi.org/10.1007/s10951-006-7042-y

Shi, Z., Huang, Z., & Shi, L. (2018). Customer order scheduling on batch processing machines with incompatible job families. *International Journal of Production Research, 56*(1–2), 795–808. https://doi.org/10.1080/00207543.2017.1401247

Wu, C. C., Liu, S. C., Zhao, C., et al. (2018). A multi-machine order scheduling with learning using the genetic algorithm and particle swarm optimization. *The Computer Journal, 61*(1), 14–31. https://doi.org/10.1093/comjnl/bxx021

Wu, C. C., Yang, T. H., Zhang, X., et al. (2019). Using heuristic and iterative greedy algorithms for the total weighted completion time order scheduling with release times. *Swarm and Evolutionary Computation, 44*, 913–926. https://doi.org/10.1016/j.swevo.2018.10.003

Wu, C. C., Bai, D., Zhang, X., et al. (2021). A robust customer order scheduling problem along with scenario-dependent component processing times and due dates. *Journal of Manufacturing Systems, 58*, 291–305. https://doi.org/10.1016/j.jmsy.2020.12.013

Wu, C. C., Gupta, J. N. D., Lin, W. C., et al. (2022). Robust scheduling of two-agent customer orders with scenario-dependent component processing times and release dates. *Mathematics, 10*(9), 1545. https://doi.org/10.3390/math10091545

Xu, J., Wu, C. C., Yin, Y., et al. (2016). An order scheduling problem with position-based learning effect. *Computers & Operations Research, 74*, 175–186. https://doi.org/10.1016/j.cor.2016.04.021

Yousefi, M., & Yusuff, R. M. (2013). Minimising earliness and tardiness penalties in single machine scheduling against common due date using imperialist competitive algorithm. *International Journal of Production Research, 51*(16), 4797–4804. https://doi.org/10.1080/00207543.2013.774475

Zhao, Z., Zhou, M., & Liu, S. (2022). Iterated greedy algorithms for flow-shop scheduling problems: A tutorial. *IEEE Transactions on Automation Science and Engineering, 19*(3), 1941–1959. https://doi.org/10.1109/TASE.2021.3062994