Rocholl, Jens; Mönch, Lars



# Metaheuristics for solving a flexible flow-shop scheduling problem with s-batching machines

International Transactions in Operational Research

**Provided in Cooperation with:**
John Wiley & Sons

*Suggested Citation:* Rocholl, Jens; Mönch, Lars (2024) : Metaheuristics for solving a flexible flow-shop scheduling problem with s-batching machines, International Transactions in Operational Research, ISSN 1475-3995, Wiley, Hoboken, NJ, Vol. 32, Iss. 1, pp. 38-68, https://doi.org/10.1111/itor.13491

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 32 (2025) 38–68
DOI: 10.1111/itor.13491

# Metaheuristics for solving a flexible flow-shop scheduling problem with s-batching machines

Jens Rocholl and Lars Mönch*

*Department of Mathematics and Computer Science, University of Hagen, 58097, Hagen, Germany*
*E-mail: jens.rocholl@fernuni-hagen.de [Rocholl]; Lars.Moench@fernuni-hagen.de [Mönch]*

Received 23 March 2023; received in revised form 14 March 2024; accepted 18 May 2024

**Abstract**

A scheduling problem for a two-stage flexible flow shop with s-batching machines motivated by processes in additive manufacturing is considered. A batch is a group of jobs that are processed together on a single machine. Each job belongs to an incompatible family. Only jobs of the same family can be batched together. A maximum batch size is given. A setup time occurs between different batches. The jobs of a batch are processed in a serial manner, that is, the processing time of a batch is the sum of the processing times of the jobs forming the batch. Batch availability is assumed. A job has a weight, a due date, and a release date. The total weighted tardiness is considered as performance measure. We establish a mixed integer linear programming formulation for this scheduling problem. For large-sized problem instances, an iterative decomposition approach (IDA) is proposed that uses a grouping genetic algorithm or an iterated local search (ILS) scheme to solve the single-stage subproblems. Moreover, an alternative ILS scheme based on a disjunctive graph representation of the problem at hand is designed. Results of computational experiments based on randomly generated problem instances demonstrate that the IDA hybridized with ILS outperforms the two other schemes.

*Keywords:* flow-shop scheduling; s-batching, decomposition; disjunctive graph; Iterated local search; grouping genetic algorithm

## 1. Introduction

Flexible flow-shop scheduling problems are important in many industrial domains such as semiconductor manufacturing, printing, food processing, and textile industries (Emmons and Vairaktarakis, 2013; Mönch et al., 2013). Since flow shop scheduling problems are typically NP-hard, metaheuristics are applied to solve them. Solution approaches are often based on stage-based decomposition approaches. We discuss a two-stage flexible flow shop with s-batch processing machines at each stage. s-batch processing means that the jobs belonging to a batch are processed in a serial manner (Webster and Baker, 1995; Potts and Kovalyov, 2000).

Such processes can be found, for instance, in additive manufacturing (AM). Fused deposition modeling (FDM) is a 3D printing process in which a work piece is built up layer by layer from plastic extruded through a heated nozzle (Gibson et al., 2021). The distance traveled by the nozzle and the number of layers determine the processing time of the part. Several parts may be placed on a build platform of a printer at the same time, thus constituting a serial batch. Painting is a common finishing process in AM, and FDM parts can have very diverse geometries (Nsengimana et al., 2019). A uniform application of paint can be achieved through the use of spray-painting robots, which move a spraying head evenly around the individual work pieces (Gleeson et al., 2022). To avoid frequent color changes, several parts are processed successively in one batch operation within the spray chamber. Incompatible families arise due to different base materials at the first stage and different colors at the second stage. The sum of the sizes of the jobs belonging to a batch must not exceed a prescribed maximum batch size given by the dimensions of the build platforms and spray chambers.

Despite its practical relevance, s-batching problems with a maximum batch size are only rarely discussed in the literature. In the present paper, we study a model problem for a two-stage flexible flow shop. We propose an iterative decomposition approach (IDA) that exploits the structure of the considered flexible flow shop. The resulting single-stage subproblems are solved by a grouping genetic algorithm (GGA) and an iterated local search (ILS) scheme. Moreover, we also propose an alternative approach for the ILS that is based on the disjunctive graph representation of the scheduling problem. For this, we extend the batch-oblivious approach proposed by Knopp et al. (2017) for p-batching to s-batching problems. In p-batching, a group of jobs is processed at the same time on a single machine (Fowler and Mönch, 2022). We demonstrate by extensive computational experiments based on randomly generated problem instances that the IDA based on ILS outperforms the one that is based on GGAs and also the ILS scheme based on the disjunctive graph representation. This paper is a considerably extended version of the conference paper by Rocholl and Mönch (2023). It includes the batch-oblivious approach as a third, alternative, approach and much more computational results.

The paper is organized as follows. In the next section, we describe and analyze the scheduling problem at hand. Related work is then discussed in Section 3. The different metaheuristic approaches are presented in Section 4. Computational results are reported and analyzed in Section 5. Finally, conclusions and future research directions are discussed in Section 6.

## 2. Problem statement and analysis

We start by introducing the scheduling problem at hand in Section 2.1. We then establish a mixed integer linear programming (MILP) formulation in Section 2.2.

### *2.1. Problem setting*

A two-stage flexible flow shop is considered. Identical s-batching machines are assumed on each stage, that is, the processing time of a batch is the sum of the processing times of the jobs that belong to the batch. Moreover, incompatible job families are assumed, that is, only jobs of the

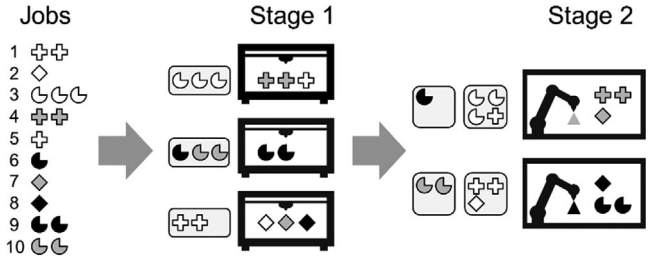



Fig. 1. Example of a problem instance.

same family can belong to a batch. A job $j = 1, \ldots, n$ has a size $s_j$, a ready time $r_j \geq 0$, a due date $d_j$, and a weight $w_j$ to express the importance of job $j$. A constant setup time $q_s$ is assumed between two batches at stage $s$. Setups can be performed by anticipating arriving jobs. A batch can only be started if all jobs belonging to the batch are ready for processing. Batch availability is assumed, that is, the jobs of a batch can only be processed further when all jobs of the batch are completed. We have a maximum batch size $B_s < \sum_{j=1}^{n} s_j$ on stage $s$, that is, the sum of the sizes of the jobs belonging to the batch does not exceed $B_s$. The performance measure to be minimized is the $TWT := \sum_{j=1}^{n} w_j (C_j - d_j)^+$ where $C_j$ is the completion time of job $j$, and we abbreviate $x^+ := \max(x, 0)$. Using the three-field notation of deterministic scheduling theory (Graham et al., 1979), the problem at hand can be stated as

$$FF2 \left| F_s, s - batch, r_j, s_j, q_s \right| TWT, \tag{1}$$

where $FF2$ indicates a flexible flow shop, $s - batch$ s-batching machines, and $F_s$ the incompatible families at stage $s$. It is shown by Cheng and Kovalyov (2001) that even the single-machine version of problem (1) with the total tardiness performance measure is NP-hard. Hence, we have to look for efficient heuristics for the flow-shop problem.

We illustrate the problem setting by means of an example. Figure 1 shows a schematic representation of a problem instance and a feasible solution.

The depicted flexible flow shop comprises three machines with a capacity of three units at the first and two machines with a capacity of four units at the second stage. Ten jobs are to be scheduled. The number of symbols indicates the size of a job. The shape of a symbol represents the family in the first stage, and its color stands for the family in the second stage. The formation of batches, machine assignments, and processing sequences resulting from a given schedule are shown. On the one hand, batches with jobs {4, 5} and {3} are processed on machine 1, {9} and {6, 10} on machine 2, and {2, 7, 8} and {1} on machine 3 at the first stage. On the other hand, batches with jobs {4, 7}, {3, 5}, and {6} are processed at machine 1 and {8, 9}, {1, 2}, and {10} at machine 2 at the second stage.

This example is enriched by concrete data for the instance shown in Fig. 1 in Table A1 of Appendix A. This includes also the resulting objective function value. Moreover, the schedule is visualized in Fig. A1 of Appendix A.

### 2.2. MILP formulation

Next, an MILP formulation for problem (1) is established. We have the following sets and indices:

$j = 1, \ldots, n$ : set of jobs,
$f = 1, \ldots, F_s$ : set of families of stage $s \in \{1, 2\}$,
$b = 1, \ldots, b_{sm}$ : set of batches on machine $m$ of stage $s \in \{1, 2\}$,
$m = 1, \ldots, m_s$ : set of machines at stage $s \in \{1, 2\}$.

Moreover, the following parameters are used in the formulation:

$B_s$ : maximum batch size of a machine at stage $s$,
$p_{fs}$ : unit processing time, that is, for a single piece, of family $f$ at stage $s$,
$d_j$ : due date of job $j$,
$q_s$ : setup time at stage $s$,
$w_j$ : weight of job $j$,
$r_j$ : ready time of job $j$,
$s_j$ : size of job $j$,
$e_{jfs}$ : $\begin{cases} 1, \text{ if job j belongs to family f at stage s} \\ 0, \text{ otherwise,} \end{cases}$.
$M$ : big number.

The following decision variables are applied in the model:

$x_{jbsm}$ : $\begin{cases} 1, \text{ if job j belongs to batch b on machine m at stage s} \\ 0, \text{ otherwise,} \end{cases}$.
$y_{bfsm}$ : $\begin{cases} 1, \text{ if batch b on machine m at stage s belongs to family f} \\ 0, \text{ otherwise,} \end{cases}$.
$C_{js}$ : completion time of job $j$ at stage $s$,
$T_j$ : tardiness of job $j$,
$a_{bsm}$ : start time of batch $b$ on machine $m$ at stage $s$.

The model itself is formulated as follows:

$$\min \sum_{j=1}^{n} w_j T_j, \tag{2}$$

subject to

$$\sum_{b=1}^{b_{sm}} \sum_{m=1}^{m_s} x_{jbsm} = 1, \; j = 1, \ldots, n, \; s \in \{1, 2\}, \tag{3}$$

$$\sum_{j=1}^{n} s_j x_{jbsm} \leq B_s,\ s \in \{1,2\},\ m = 1,\ldots,m_s,\ b = 1,\ldots,b_{sm}, \tag{4}$$

$$\sum_{f=1}^{F_s} y_{bfsm} = 1,\ s \in \{1,2\},\ m = 1,\ldots,m_s,\ b = 1,\ldots,b_{sm}, \tag{5}$$

$$e_{jfs} x_{jbsm} \leq y_{bfsm},\ j = 1,\ldots,n,\ s \in \{1,2\},\ f = 1,\ldots,F_s,\ m = 1,\ldots,m_s,\ b = 1,\ldots,b_{sm}, \tag{6}$$

$$r_j x_{jb1m} \leq a_{b1m},\ j = 1,\ldots,n,\ b = 1,\ldots,b_{1m},\ m = 1,\ldots,m_1, \tag{7}$$

$$q_1 \leq a_{b1m},\ j = 1,\ldots,n,\ b = 1,\ldots,b_{1m},\ m = 1,\ldots,m_1, \tag{8}$$

$$a_{bsm} + q_s + \sum_{j=1}^{n}\sum_{f=1}^{F_s} e_{jfs} p_{fs} s_j x_{jbsm} \leq a_{b+1,s,m},\ s \in \{1,2\},\ b = 1,\ldots,b_{sm},\ m = 1,\ldots,m_s, \tag{9}$$

$$a_{bsm} + \sum_{k=1}^{n}\sum_{f=1}^{F_s} e_{kfs} p_{fs} s_k x_{kbsm} \leq C_{js} + M\left(1 - x_{jbsm}\right),\ j = 1,\ldots,n, s \in \{1,2\},\ m = 1,\ldots,m_s, b = 1,\ldots,b_{sm}, \tag{10}$$

$$C_{j1} \leq M\left(1 - x_{jb2m}\right) + a_{b2m},\ j = 1,\ldots,n,\ m = 1,\ldots,m_s, b = 1,\ldots,b_{sm}, \tag{11}$$

$$C_{j2} - T_j \leq d_j,\ j = 1,\ldots,n, \tag{12}$$

$$x_{jbsm}, y_{bfms} \in \{0,1\},\ C_{js}, T_j, a_{bsm} \geq 0,\ j = 1,\ldots,n, s \in \{1,2\},\ f = 1,\ldots,F_s,\ b = 1,\ldots,b_{sm}, m = 1,\ldots,m_s. \tag{13}$$

The objective function (2) to be minimized is the total weighted tardiness (TWT). Constraint set (3) ensures that each job at each stage is assigned to exactly one batch. Constraint (4) enforces that the maximum batch size is respected for each formed batch. Constraint set (5) makes sure that exactly one family is assigned to a batch. The family of the jobs belonging to a batch and the family assigned to a batch is the same. This is modeled by constraint set (6). Constraint set (7) relates the starting time of the first batch on a given machine to the ready time of the jobs that belong to the batch. Constraint (8) makes sure that setup times are considered at the first stage. The starting times of adjacent batches on a given machine are modeled by constraint set (9). Constraint set (10) computes the completion time of jobs based on the completion time of the related batch. The ready time of a batch at the second stage is not earlier than the completion time of any job of the batch at the first stage. This is expressed by constraint (11). Constraint set (12) linearizes the tardiness. The domains of the decision variables are modeled by (13).

Since problem (1) is NP-hard, we cannot expect that medium- or large-sized instances of the MILP model (2)–(13) can be solved in a reasonable amount of computing time using a commercial solver. However, the model can be used to check the correctness of the coded metaheuristics.

## 3. Related work

A survey for batching including s-batching is provided by Potts and Kovalyov (2000). A single-machine s-batching problem with a minimum and a maximum batch size and batch availability and the total completion time measure is studied in Mosheiov and Oron (2008). Efficient optimal solution procedures are designed. Dynamic programming algorithms are proposed for a single-machine scheduling problem with s-batching, minimum and maximum batch sizes, and batch availability in Chrétienne et al. (2011). Here, the performance measure is the sum of the tardiness and the setup costs. A tabu search approach for a single-machine s-batching problem with a maximum batch size and batch availability and the TWT measure is discussed by Suppiah and Omar (2014). Polynomial-time approaches to find Pareto-optimal schedules are established. A parallel-machine scheduling problem with s-batching, batch availability, family sequence-dependent setup times, and the total weighted completion time performance measure is studied in Shen et al. (2014). Variable neighborhood search (VNS) procedures that iterate between batch formation and sequencing are proposed. A single-machine s-batching problem with a minimum and a maximum batch size and batch availability where the maximum lateness and makespan are the performance measures is analyzed by He et al. (2015). A parallel-machine scheduling problem with s-batching, incompatible job families, unequal job sizes, and a maximum batch size is studied by Gahm et al. (2022). The TWT performance measure is considered. Heuristics hybridized with local search (LS) are proposed. A multi-start construction heuristic combined with machine learning techniques is proposed for a similar scheduling problem by Uzunoglu et al. (2023). All the discussed scheduling problems are for single- or parallel-machine environments, metaheuristic approaches are rarely used. Hence, to the best of our knowledge, scheduling problem (1) is not discussed in the literature so far.

The most pertinent papers for the scheduling problem (1) are Tan et al. (2018) and Knopp et al. (2017). A stage-wise IDA for a two-stage flexible flow shop with p-batching and the TWT measure is designed by Tan et al. (2018). This IDA will be applied in the present paper to obtain parallel-machine subproblems for which metaheuristic approaches are proposed. The disjunctive graph representation for scheduling problems with p-batching is typically based on introducing additional nodes and arcs for the batching decisions (cf. Ovacik and Uzsoy, 1997). This often leads to a huge number of additional nodes and arcs. This limitation is reduced in Knopp et al. (2017) by proposing a batch-oblivious approach that makes the additional nodes and arcs for batching decisions obsolete. In the present paper, we extend the batch-oblivious approach toward s-batching situations. To the best of our knowledge, disjunctive graphs are not applied so far to tackle flow-shop or job-shop scheduling problems with s-batching machines.

The contribution of the present paper can be summarized as follows:

1. We analyze a two-stage flexible flow-shop scheduling problem with s-batching machines.
2. An IDA is proposed where the resulting subproblems are solved by a GGA and an ILS.

3. Moreover, we examine the question whether the disjunctive graph representation offers some advantage for the scheduling problem at hand. For this, we extend the batch-oblivious approach for p-batching to s-batching situations. We propose several ILS variants based on the disjunctive graph representation.

## 4. Metaheuristics for the scheduling problem

We start by discussing the main ideas of the IDA in Section 4.1. We then discuss solution procedures for the subproblems per stage in Section 4.2. The batch-oblivious approach for s-batching machines is presented in Section 4.3.

### *4.1. IDA*

We adapt the IDA framework proposed by Tan et al. (2018) to the characteristics of the problem at hand, namely, the s-batch processing. Different metaheuristics are incorporated to tackle the derived subproblems. The idea of the IDA consists in first decomposing the problem by stage and then iteratively solving the resulting single-stage subproblems one after another. Each solution of a single subproblem provides parameters for the consecutive one, that is, completion times of jobs in a solution to the first-stage subproblem serve as release dates to the second-stage subproblem of the same iteration. Similarly, the starting times of the batches the jobs are assigned to in a solution to the second-stage subproblem are the basis for computing internal due dates of jobs of the first-stage subproblem of the next iteration. By computing high-quality solutions given these parameters, the intention of the approach is to successively obtain better solutions with regard to the overall problem. For the first iteration, as no solution for a subproblem is available yet, reasonable internal due dates are estimated by

$$d_{j1}^{(1)} := 1/2\big(r_j + p_{j1}s_j + \ d_{j2} - p_{j2}s_j\big). \tag{14}$$

Recall that $p_{js}$ is used to denote the unit processing time of the family of job $j$ at stage $s$. The right-hand side of Equation (14) is the point in time in the middle between the earliest possible completion time of a job's first operation and the latest time its second operation must be started to be completed on time. The internal due date can therefore be seen as a compromise that allows for both operations the same slack. Obviously, it is just a crude estimate as capacity, setup, and batching are neglected, but it is supposed to perform well enough as a starting point for the heuristic. In subsequent iterations, the internal due dates are computed by

$$d_{j1}^{(g)} := (1-\alpha)\, r_{j2}^{(g-1)} + \alpha\left(d_{j2} - b_{j2}^{(g-1)} - W_{j2}^{(g-1)}\right) + \beta W_{j2}^{(g-1)}, \tag{15}$$

where $b_{js}^{(g)}$ is the processing time of the batch the job $j$ was assigned to at stage $s$, and $W_{js}^{(g)}$ is the waiting time of job $j$ at stage $s$ in iteration $g$. The waiting time of a job is the difference of its actual start of processing in a schedule and its availability, that is, we have $W_{js}^{(g)} = s_{js}^{(g)} - r_{js}^{(g)}$ where $s_{js}^{(g)}$ is

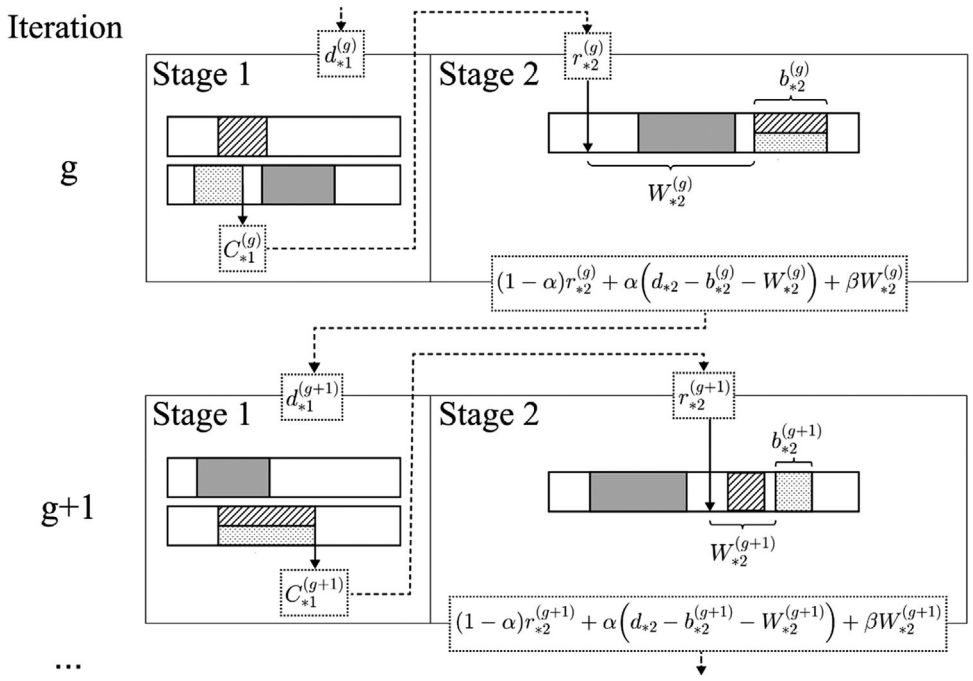



Fig. 2. Iterative decomposition approach (IDA) scheme (two consecutive iterations).

the start time of job $j$ at stage $s$ in iteration $g$. Rearranging the terms of the right-hand side of Equation (15) yields

$$d_{j1}^{(g)} := (1-\alpha)\, s_{j2}^{(g-1)} + \alpha\left(d_{j2} - b_{j2}^{(g-1)}\right) + (\beta - 1)\; W_{j2}^{(g-1)}, \tag{16}$$

that is, three terms contribute to the internal due dates of the jobs in the first stage. The weights of actual start times of an operation and slack in the previous iteration are controlled by a parameter $\alpha \geq 0$. To compute the slack, we need to consider the actual processing time $b_{js}^{(g)}$, which depends on the case of s-batching on all jobs belonging to the batch. The parameter $\beta \geq 0$ is used to control the weight of the waiting time in the schedule associated with the previous iteration. Internal release dates for the second stage are set in a straightforward manner by $r_{j2}^{(g)} := C_{j1}^{(g)}$. The approach is illustrated by means of a schematic representation of an example shown in Fig. 2.

The figure shows two iterations of the procedure by highlighting a single job labeled by an asterisk symbol *. Assume there are two machines at the first stage and a single one at the second stage. At the first stage of iteration $g$, a schedule for that subproblem, represented by a Gantt chart in the figure, is obtained by means of the procedure therefore designed. The schedule considers internal due dates either computed in iteration $g-1$ or estimated as described above if $g = 1$. From that schedule, we derive completion times to be considered as release dates for the second stage. Again, a schedule is obtained at the second stage from which waiting times and actual processing times of batches can be taken to compute internal due dates for the first stage in iteration $g+1$.

### 4.2. Subproblem solution procedures

#### 4.2.1. Overall approach

A stage-based decomposition of problem (1) yields two subproblems of the type:

$$Pm\left|F_s, s-batch, r_j, s_j, q_s\right| TWT. \tag{17}$$

where $Pm$ denotes identical parallel machines. The subproblem remains hard to solve because of the parallel machine environment, the batching characteristic, job release dates, and the TWT performance measure since even the single-machine version of problem (17) is known to be NP-hard due to Cheng and Kovalyov (2001). Therefore, it is necessary to tackle it with heuristic or metaheuristic methods to solve large-sized problem instances within a reasonable amount of computing time. In the remaining part of this subsection, two different approaches are proposed, namely, a population-based approach and a neighborhood search approach.

#### 4.2.2. GGA for a parallel-machine setting

Falkenauer (1998) proposes GGAs to overcome the shortcomings of traditional encoding schemes of GAs when grouping decisions are made. If genes represent jobs or positions of jobs, genetic operations tend to disrupt the formation of groups of chromosomes to a large extent and thus impair the effectiveness of the propagation of favorable grouping decisions. With a GGA, genes represent the actual grouping entities, that is, batches, along with their contents, the assigned jobs. Coupled with specifically designed genetic operators, the GGA representation is better suited to facilitate the inheritance of favorable groups (Falkenauer, 1996). GGAs are successfully applied to different scheduling problems with p-batching or multiple orders per job characteristics (for instance, see Sobeyko and Mönch, 2011; Sobeyko and Mönch, 2015; Rocholl et al., 2020; among others).

In the proposed GGA, a two-piece random key is complementary and added to encode the machine assignment and the sequence of the jobs on a single machine. The first part of the key, that is, the greatest integer less than or equal to the key, is an integer from the range $[1, \ldots, m]$ indicating the machine where a batch is scheduled on. The sorting order over the second real number part of the keys, that is, the difference of the key and the integer part, of all genes assigned to the same machine defines the processing sequence. For the first generation, all chromosomes are initialized randomly.

The crossover operation is designed to largely preserve the contents of a batch when it is passed on to an offspring. The roulette wheel selection method (Goldberg, 1989; Michalewicz, 1996) is used to select two chromosomes as first and second parent for a crossover operation. A two-point crossover is implemented. An offspring is created by first copying the genes outside the randomly set demarcation from the first parents. The offspring inherits the genes delimited by the crossover points from the second parent. To ensure that the offspring can be decoded to a feasible solution, possibly duplicate jobs need to be deleted and/or missing jobs need to be reinserted. In the proposed GGA, duplicate jobs are always deleted from the batches inherited from the first parent. Missing jobs are reinserted in random order into suitable batches processed as early as possible but not started before the job's release date. The crossover operation is schematically shown in Fig. 3. Two chromosomes each containing 10 jobs that belong to four and five batches

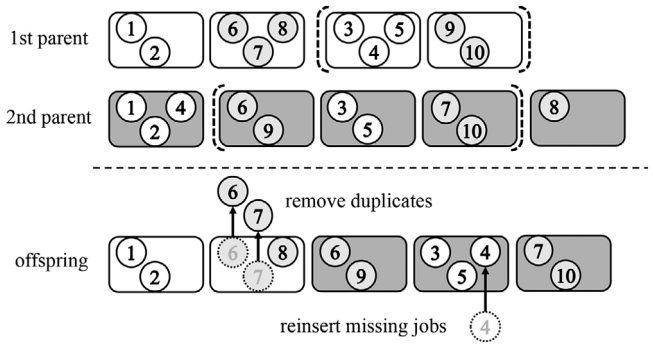



Fig. 3. Grouping genetic algorithm crossover operation.

are depicted in Fig. 3. It can be seen that jobs 6 and 7 are removed from the second batch of the child chromosome, whereas job 4 is reinserted into the fourth batch of the child chromosome.

The mutation operator is designed to randomly reinitialize the key of one randomly selected gene. A steady-state GA with overlapping populations is used, that is, the individuals of the population with the lowest fitness are replaced according to a prescribed replacement rate.

LS is incorporated to further guide the search toward regions of the search space containing high-quality solutions. To obtain a schedule for an instance of problem (17), three decisions must be made. Batches have to be formed from the operations assigned to one of the machines, and a processing sequence of batches must be established. We seek to design the LS to address all of these decisions by choosing five neighborhood structures, defined by the following types of moves:

1. Insert operation: Select one operation randomly and insert it into a different, randomly selected batch with sufficient capacity and operations of the same family or into an entirely empty batch.
2. Swap operation: Randomly select and exchange two operations of jobs belonging to batches of the same family, if sufficient capacity is available.
3. Insert Batch: Move a randomly selected batch to a randomly selected machine and position.
4. Swap Batch: Swap the positions of two randomly selected batches by considering all positions at all machines.
5. Split Batch: A randomly selected batch is split into two batches. The distribution of operations to the two batches is again determined randomly.

The first two types of moves and the last one alter the batch formation, whereas the third and fourth move types can lead to changes in both the batch assignment and the processing sequence. These move types are already applied successfully to p-batching problems (Bilyk et al., 2014).

After each move, a consecutive workload balancing step is performed before the impact on the objective function value is evaluated, and a move is only accepted if it leads to an improvement. Therefore, the batch with the highest completion time is determined. If the batch cannot be assigned to a different machine so that it is completed earlier, the workload is considered to be balanced. Otherwise, the batch is assigned to the machine that becomes available first. The procedure is repeated until the workload is balanced. The workload balancing might be necessary because in the s-batching setting, the above-mentioned moves usually alter the processing times of

the batches and thus can lead to schedules with a workload unevenly distributed across the parallel machines. Thus, a move without workload balancing can be wrongly assessed to have a negative impact, and therefore it can mistakenly not be accepted.

Ten moves of each type are executed in the given order. The LS scheme is thus limited to keep the computing time requirement of the GGA low to allow for multiple iterations of the IDA. Preliminary experimentation with a small number of problem instances demonstrates that this limitation is reasonable. The IDA approach with the integrated GGA heuristic is abbreviated as IDA-GGA.

#### *4.2.3. ILS for a parallel-machine setting*

ILS is a metaheuristic that is successfully applied to solve different classes of scheduling problems (Lourenço et al., 2010; Stützle and Ruiz, 2018). It is well-known that neighborhood search approaches such as VNS or ILS often perform well for hard combinatorial optimization problems (cf. Voß and Woodruff, 2006).

The main idea of ILS consists in continuously moving along a sequence of solutions, iteratively steering the search toward a local optima and escaping them to find different optima with the intention to gradually move closer toward a global optimum. In the proposed ILS scheme, first, an initial solution is obtained by simply assigning all jobs to individual batches, that is, only a single job belongs to each batch, and assigning and sequencing of these batches using list scheduling and the apparent tardiness cost (ATC) dispatching rule (Vepsalainen and Morton, 1987). The ATC index of job $j$ for stage $s$ decisions is given by:

$$I_j(t) := w_j / p_{js} e^{\left(-\left(d_{js} - p_{js} s_j - \left(r_{js} - t\right)^+\right)^+ / \kappa \bar{p}\right)}. \tag{18}$$

where $\kappa$ is a look-ahead parameter, and $\bar{p}$ is the average processing time of all unscheduled jobs. Moreover, $d_{js}$ and $r_{js}$ are the due date and ready time of job $j$ at stage $s$, and $t$ is the time where the decision is made. The values of the parameter $\kappa$ are taken from an equidistant grid, and the schedule with the smallest objective function value is used for initialization purposes.

One iteration of the ILS procedure consists of a LS phase that transforms the current solution into one reaching a local minimum and a perturbation phase that ensures to escape from it to continue searching in different regions of the search space. The LS scheme is formed by a randomized variable neighborhood descent (RVND) (Hansen and Mladenovic, 2001; Hansen et al., 2010), similar to the approach proposed by Queiroga et al. (2021) for a p-batching problem. The five neighborhood structures defined by the following moves are applied.

1. Insert operation: Any operation is considered to be placed into any different batch that is started earlier. Only batches composed of operations of the same family with sufficient available capacity need to be considered.
2. Swap operations: Any two operations belonging to the same family are considered to switch places.
3. Insert batch: Any batch is considered to be inserted into any position at any of the parallel machines.
4. Swap batches: Any two batches are considered to switch positions in the schedule.
5. Split batch: Any batch is considered to be split into two batches.

The RVND algorithm randomizes the order of these structures and performs a move by applying a best-fit strategy, that is, the move that leads to the largest decrease of the objective function value is performed, if there is at least one move that actually leads to a better solution. If such a move is identified then the loop starts all over again, randomizing the order of neighborhood structures and starting with a greedy search in the first one. If within one neighborhood no better solution can be obtained, then the search continues with the next neighborhood structure. Hence, the LS phase terminates when no better solution can be found using any of the neighborhood structures.

In the following perturbation phase, two random moves are performed. The type of each move is randomly chosen from the set of the types of insert operation, swap operation, or split batch with equal probability. The LS and perturbation phases constitute a single iteration of the procedure. The pseudocode of the proposed RVND procedure is provided in Appendix B of the paper for the sake of completeness. The combination of the IDA and the ILS scheme is abbreviated by IDA-ILS in the rest of this paper.

### *4.3. Batch-oblivious approach*

Scheduling problems with batching characteristic can be represented using a disjunctive graph model (Ovacik and Uzsoy, 1997). This modeling approach is widely used in the literature for p-batching problems (Fowler and Mönch, 2022) and also for flow shops (cf., for instance, Koulamas, 1998; Demirkol and Uzsoy, 2000; Yang et al., 2000; and for representations of flow shop scheduling problems, Panwalkar and Koulamas, 2019). We adapt the batch-oblivious approach proposed by Knopp et al. (2017) for p-batching problems to the problem at hand including s-batching and design an ILS algorithm, which alters the graph structure to gradually find schedules with a lower TWT value. In the following two subsections, the graph model and the heuristic approach are described in some detail.

#### *4.3.1. Disjunctive graph representation*

A disjunctive graph $G = (V, E)$ with a set of vertices or nodes $V$ and a set of edges or arcs $E$ is considered. Nodes of the graph represent the operations of the scheduling problem. The nodes associated with operations of the same job are connected with directed arcs according to the technological precedence constraints implied by the routes of the jobs. A pair of two oppositely directed arcs between any two nodes representing operations of one stage is introduced to model the possible sequencing decisions. In the following, the aforementioned subset of arcs will be referred to as route arcs while the latter are called sequence arcs. For each of the machines available at the first stage, a single artificial source node is added to the set $V$. The TWT measure requires to consider the individual completion times of all jobs. Therefore, it is also necessary to introduce a sink node for each job, which is connected by an incoming route arc coming from the second-stage node of the job. Processing times of operations are represented by outgoing arcs. The outgoing arcs of the source nodes are weighted with the maximum of the release dates and the setup times of the jobs they are directed to. The disjunctive graph is exemplified for problem (1) with three jobs and a single machine at the first stage in Fig. 4. The notation $j.s$, $s \in \{1, 2, *\}$ is used to indicate that operation $s$, or the sink * of job $j$ is associated with a node of the graph.

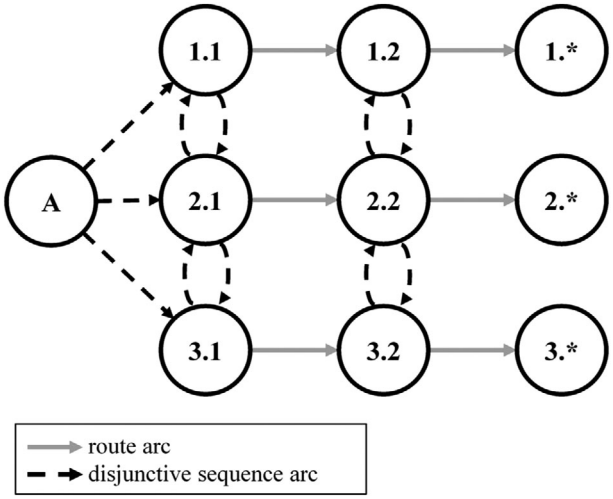



Fig. 4. Disjunctive graph example.

From the disjunctive graph, we can derive a conjunctive graph representing a feasible schedule by fixing at most one of the sequence arcs of each pair and discarding the others while additional constraints are satisfied. For at least one source node, exactly one outgoing sequence arc needs to be fixed, while the others are discarded. At most, one outgoing arc may be fixed for each source node. A weight equal to the processing time of the operation associated with the preceding node plus the setup time at the related stage is assigned to the fixed sequence arcs. At any time, it needs to be ensured that there are no cycles in the graph. Moreover, for each subset of nodes representing the operations of a single stage, there must be no more distinct paths formed of sequence arcs than there are machines available at that stage. That is because each of these directed paths represents the sequence of operations processed at a single machine. A conjunctive graph obtained in such a way represents a schedule where the completion time of a job is equal to the weight of the longest path to its associated sink node. A feasible schedule is represented by a directed acyclic graph. The computation of the longest paths to all nodes can be achieved in linear time based on a topological ordering of the nodes. For each node, the length of the longest path is equal to the length of the longest path to any of its preceding nodes plus the weight of the connecting arc. The topological ordering ensures that the longest paths of preceding nodes are already computed before. When the values are saved for each node, a single pass through the ordered set of nodes is sufficient to compute all the longest paths in the graph.

We adapt the batch-oblivious approach proposed by Knopp et al. (2017) to s-batching decisions. The essence of this approach consists in avoiding dedicated nodes representing batches. Instead, the batching of operations is represented by assigning a weight of zero to all connecting sequence arcs between operations belonging to a batch. We refer to arcs with a weight of zero in the remainder of this paper as batching arcs. A path solely consisting of batching arcs is called a batching chain.

Batching imposes additional constraints to be met by conjunctive graphs. The sum of the sizes of operations related to the nodes of a batching chain must not exceed the given batch capacity at the associated stage. Furthermore, the family those operations belong to must be the same.

As in Knopp et al. (2017), we use the notation $r(v)$ to refer to the route successor of a node $v$ and $m(v)$ to refer to its machine successor. Its predecessors are denoted $r^{-1}(v)$ and $m^{-1}(v)$, respectively. Given a conjunctive graph, the start time $S_v$ of the operation associated with $v$ can be derived by computing the longest path to $v$. As each node in the conjunctive graph has an in-degree of at most two, with one incoming route arc and at most one incoming sequence arc, the longest path can be recursively computed as

$$S_v = \max\left(S_{r^{-1}(v)} + l_{r^{-1}(v),v'} S_{m^{-1}(v)} + l_{m^{-1}(v),v}\right). \tag{19}$$

where $l_{u,v}$ is the weight of the arc between $u$ and $v$.

With the batch-oblivious approach, batching of operations $v$ and $m(v)$ is represented by a sequence arc with a weight $l_{v,m(v)} = 0$ connecting the two corresponding nodes. Because in a feasible solution of problem (1), all operations belonging to the same batch are started at the same time, it must be ensured that for the start times, we have $S_v = S_{m(v)}$ if $l_{v,m(v)} = 0$. The precedence of nodes modeled by the structure of a conjunctive graph only guarantees $S_v \leq S_{m(v)}$. To ensure that also $S_v \geq S_{m(v)}$ holds, we derive

$$l_{v,m(v)} = 0 \;\Rightarrow\; S_v \geq S_{r^{-1}(m(v))} + l_{r^{-1}(m(v)),m(v)} \tag{20}$$

from Equation (19). Following Knopp et al. (2017), we introduce an invariant to be fulfilled by all nodes $v \in V$ and $w \in V$ with $w = r^{-1}(m(v))$. If two adjacent operations processed at the same machine do not belong to a batch, then $l_{v,m(v)} > 0$ and $S_v < S_{m(v)}$ hold. Contrary to the case of p-batching, the s-batching restriction may require the consideration of additional nodes to define the value of $l_{v,m(v)}$. Let $B_v$ be the set of operations belonging to the same batch as $v$, corresponding to the set of nodes comprising a batching chain containing $v$. Then $l_{v,m(v)} = q_s + \sum_{u \in B_s} p_u s_u$ holds. Consequently, the invariant

$$\left(l_{v,m(v)} = 0 \;\wedge\; S_v \geq S_w + l_{w,m(v)}\right) \vee \left(l_{v,m(v)} = \sum_{j \in B_v} p_j s_j + q_s\right) \tag{21}$$

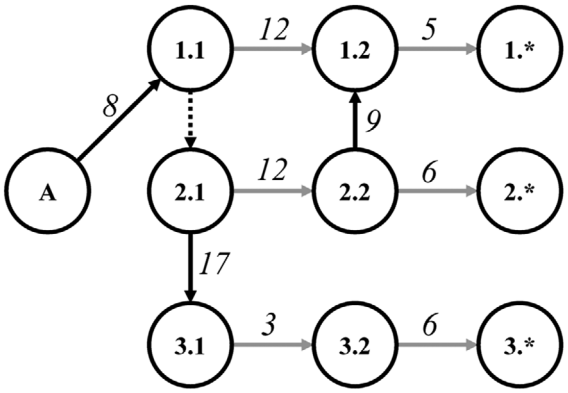
needs to be fulfilled.

A similar requirement is implied by the s-batching mode in combination with batch availability. As the processing time of a batch is the sum of the processing times of assigned operations and all these operations are completed at the same time, the constraint

$$S_v + \sum_{j \in B_v} p_j s_j \;\leq S_{r(v)} \tag{22}$$

must be fulfilled.

Other than in the case of p-batching (see Knopp et al., 2017), properties (21) and (22) do not allow to make batching decisions for s-batching problems dynamically in a single pass through the set of nodes sorted in topological order because changes of the set $B_v$ affect the weight of outgoing arcs of nodes already settled. Instead, the changes of batching decisions in a conjunctive graph require a traversal along the batching chain and the adaptation of the weight of both sequence and route arcs. The conjunctive graph model is illustrated by means of an example in Fig. 5.




| | Job 1 | Job 2 | Job 3 |
|---|---|---|---|
| $r$ | 8 | 5 | 10 |
| $p$ at stage 1 | 5 | 7 | 3 |
| $p$ at stage 2 | 5 | 6 | 6 |

Fig. 5. Conjunctive graph example.

The graph represents a schedule for problem (1) with a single machine at the first stage and two machines at the second stage. A batch containing the first operations of jobs 1 and 2 is processed before the first operation of job 3 at the first stage. At the second stage, the second operation of job 2 is processed before the second operation of job 1 on one machine. The second operation of job 3 is processed on the second machine of this stage. The length of the longest paths to the sink node of the first job, that is, its completion time, is 34. The path itself is (A, 1.1, 2.1, 2.2, 1.2, 1.*). The completion times of job 2 and 3 are 26 and 34, respectively.

### *4.3.2. ILS for a flow-shop setting*

An initial schedule is obtained by list scheduling. For the first stage, jobs are assigned to batches consisting of a single job and sorted with respect to the index (18) of the ATC dispatching rule. These batches are then assigned to the available machines by list scheduling. The same procedure is applied to the second stage where completion times of the former one serve as release dates. The look-ahead parameter $\kappa$ is taken from an equidistant grid for both stages. The schedule with the lowest TWT value is used for constructing the corresponding conjunctive graph.

Neighborhood structures for exploration during the LS are designed by four different moves. A direct change of the batch formation is achieved by two moves called SPLIT and MERGE. With the batch-oblivious approach, such moves do not alter the structure of the graph. Instead, only the weights of arcs need to be changed. Additional moves are required to address the batch assignment and sequencing decisions.

Paulli (1995) proposes a procedure to move a node to a different position within the graph for a flexible manufacturing system. Dauzère-Pérès and Paulli (1997) show that a move of this type cannot introduce a cycle and therefore not render a feasible solution infeasible for the case of a

multiprocessor job-shop. In the following, we apply the same arguments to show that moving a node in a conjunctive graph representing a flexible flow shop with s-batching machines cannot introduce a cycle, and therefore cannot lead to an infeasible solution. Moving a node $v$ between the nodes $u$ and $w$ is accomplished with two steps. First, the sequence arcs $(m^{-1}(v),\ v)$ and $(v,\ m(v))$ are replaced by the arc $(m^{-1}(v),\ m(v))$. Second, the arc $(u,\ w)$ is replaced by the arcs $(u,\ v)$ and $(v,\ w)$. There are only two situations in which this move can introduce a cycle:

1. There is a path from $w$ to $r^{-1}(v)$ in the original graph.
2. There is a path from $r(v)$ to $u$ in the original graph.

In a flow-shop setting, all jobs follow the same route, and the sets of machines dedicated to operations of different stages are disjoint. Consequently, only operations of the same stage can be processed at the same machine. The nodes $v$, $u$, and $w$ thus represent operations of different jobs that are to be carried out at the same stage of the flow shop. If $w$ is processed at the same stage as $v$, and $r^{-1}(v)$ at the earlier stage, then there cannot be a path from $w$ to $r^{-1}(v)$, as there are no route arcs between nodes of different jobs and no sequence arcs between nodes of different stages. Analogously, there cannot be a path from $r(v)$ to $u$, and consequently neither of the two scenarios is valid. The described situation is illustrated in Fig. 6.

This reassignment technique of Paulli (1995) appears to fit the purpose of simultaneously addressing the batch formation, assignment, and sequencing decisions. However, the impact of moves of this type on the batch formation requires a more elaborated treatment, compared to the original neighborhood structure due to Dauzère-Pérès and Paulli (1997), which is given below. The move types constituting the neighborhood structures to be exploited with LS are now described in more detail. For the sake of better readability, the notion of predecessor and successor will be used to refer to the nodes connected via incoming and outgoing sequence arcs.

The proposed four move types are as follows:

1. SPLIT: Any batching arc between two nodes $u$ and $v$ is considered. To model the split of a batch, the processing times of the two resulting batches must be calculated. The set of nodes belonging to the first batch can be recursively determined by

$$\overleftarrow{B}(u) := \begin{cases} u, & \text{if } l_{m^{-1}(u),\ u} > 0 \\ u \cup \overleftarrow{B}\left(m^{-1}(u)\right), & \text{otherwise,} \end{cases} \tag{23}$$

and the set of nodes belonging to the second batch is

$$\overrightarrow{B}(v) := \begin{cases} v, & \text{if } l_{v,m(v)} > 0 \\ v \cup \overrightarrow{B}(m(v)), & \text{otherwise.} \end{cases} \tag{24}$$

A weight of $\sum_{j \in \overleftarrow{B}(u)} p_j s_j$ is assigned to all outgoing route arcs of nodes $u' \in \overleftarrow{B}(u)$, and $\sum_{j \in \overrightarrow{B}(v)} p_j s_j$ to outgoing route arcs of nodes $v' \in \overrightarrow{B}(v)$. For possible outgoing sequence arcs the setup time is added.

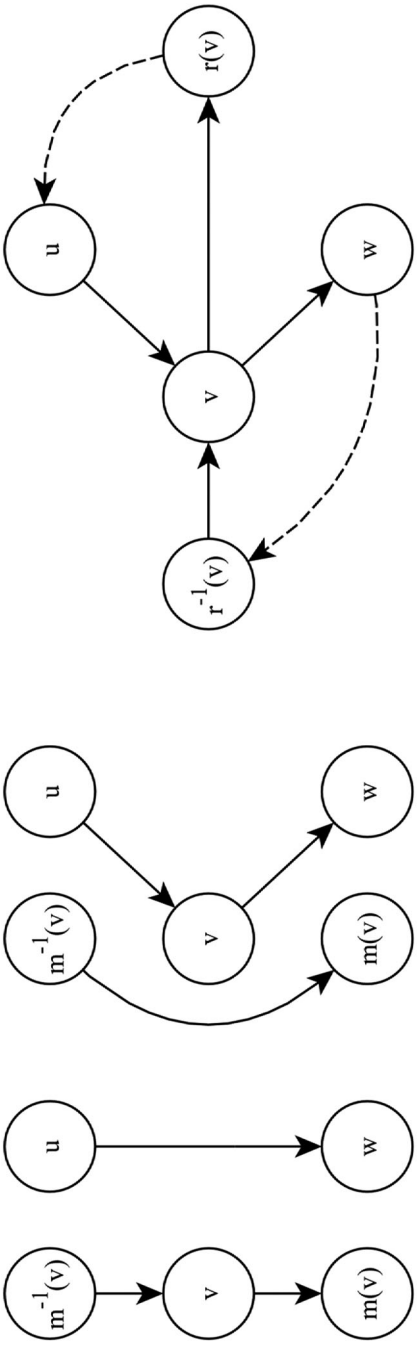



Fig. 6. Cycle avoidance after insert move.

2 MERGE: Any sequence arc between two nodes $u$ and $v$ with a weight $l_{u,v} > 0$ is considered. The operation can only be performed if both $u$ and $v$ belong to the same family and if $\sum_{j \in \overleftarrow{B}(u) \cup \overrightarrow{B}(v)} s_j \leq B$ holds. The weights of outgoing route arcs of all nodes of the combined set are set to the processing time of the resulting batch. For the weight of a possible outgoing sequence arc, the setup time is added.
3 INSERT: Any node $v$ is considered to be moved between any two nodes $u$ and $w = m(u)$ where $v$, $u$, and $w$ represent operations of the same stage. The move requires replacements of sequence arcs as described above. Each of the nodes may belong to a batch before the move is performed, and it has to be defined how to proceed in that case. As the batch formation is already addressed by the SPLIT and MERGE moves, the INSERT step is designed to largely maintain previous batching decisions. First, SPLIT operations are performed on possible batching arcs going into or out of $v$. If $m^{-1}(v)$, $v$, and $m(v)$ belong to a batching chain before the move, then a MERGE operation is performed on $m^{-1}(v)$ and $m(v)$ after the relocation of $v$. If $u$ and $w$ are connected by a batching arc before the move, then MERGE steps are performed after the insertion of $v$ first on $u$ and $v$ and afterward on $v$ and $w$. Recall that a MERGE operation includes checks on the compatibility and capacity constraints. Consequently, after the move all of the nodes, $u$ and $v$, $v$, and $w$ or none of them can belong to one batch. The operation may produce a graph where the number of distinct groups of nodes connected by machine arcs is smaller than the number of available machines at the respective stage. In that case, the most critical node $v_{crit}$ is isolated by replacing the arc $(m^{-1}(v_{crit}), v_{crit})$ and $(v_{crit}, m(v_{crit}))$ by $(m^{-1}(v_{crit}), m(v_{crit}))$. The criticality of a node is the sum of the TWT values of all connected sink nodes.
4 SWAP: Any two nodes $v$ and $w$ representing operations of the same stage are considered. The move comprises two INSERT operations where first $v$ is inserted between $m^{-1}(w)$ and $w$ and then $w$ is inserted between $m^{-1}(v)$ and $m_{pre}(v)$. Here, $m_{pre}(v)$ indicates the successor of $v$ before the execution of the move.

With the exception of the different neighborhood structures, the LS is designed as an RVND scheme corresponding to the procedure described in Section 4.2.3.

Each perturbation phase comprises two random steps that are of the type INSERT with a probability of 0.7 or of the types SPLIT or MERGE with each a probability of 0.15. The INSERT move is preferred as it is more disruptive and can as well impact the batch formation, machine assignment, and sequence.

To assess the impact of a move, it is sufficient to recalculate the weights of those arcs which lie on a path with at least one of the directly affected nodes or arcs. This limitation to directly and indirectly affected nodes can reduce the computational cost. However, in a two-stage flow shop, each node of a first-stage operation is always connected to at least its corresponding second-stage node and thus possibly to additional immediately connected nodes. The potential reduction of affected nodes would be higher on higher stages, but in the present two-stage flow shop, it may be rather small. Moreover, the lower the number of machines available and therewith the number of paths, the higher the average number of nodes affected by an LS move. The s-batch processing mode also requires additional feasibility checks and calculations. For these reasons, we do not expect a large benefit using the disjunctive graph for problem (1). However, the approach can be easily scaled to scheduling problems with more stages where it may possibly prove advantageous. We refer to the

resulting approach as ILS-DG in the remainder of this paper where the abbreviation DG is used for disjunctive graph.

Moreover, a modified version of the procedure is designed for which the described ILS loop is executed repeatedly, each time starting from a different initial solution. For the first start, the initial solution is obtained using the list scheduling approach combined with the ATC dispatching rule as described above. For consecutive starts, the initial solution is obtained by first assigning each job to an individual batch and then applying list scheduling with a random ordering of batches at both stages to obtain a schedule. This multi-start version is abbreviated by ILS-DG-MS where MS stands for multi start. The pseudocode of the ILS-DG-MS procedure is summarized in Appendix C of the present paper.

## 5. Computational experiments

The design of experiments is discussed in Section 5.1. Moreover, the applied parameter settings and implementation details are described in Section 5.2. Computational results are presented and analyzed in Section 5.3.

### *5.1. Design of experiments*

The performance of the proposed algorithms is assessed based on randomly generated problem instances. We expect that the performance of the heuristics depends on the number of jobs per family, the number of incompatible families, the distribution of workload capacity across the stages, and the release dates of the jobs. Job sizes are set according to $s_j \sim DU[1,\ 8]$ where $X \sim DU[a, b]$ denotes a discrete uniform random variable from the interval [a, b]. Batch capacities of 15 and 30 pieces are considered at both stages, which is considering the average job sizes, equivalent to an average maximum batch size of $\bar{B}_s \in \{3, 6\}$ operations per batch. The setup time is chosen as $q_s := \lfloor (s_j^{\max} - s_j^{\min}) p_s^{\max}/4 \rfloor$ where $s_j^{\max}$ and $s_j^{\min}$ are the maximum and minimum job sizes, respectively. Moreover, $p_s^{\max}$ is the maximum unit processing time among the families.

Three different bottleneck scenarios are examined. The number of machines at each stage is computed based on the given total number of machines using the bottleneck configuration (BC) algorithm, described by Tan et al. (2018), and adapted to the case of serial batching. The workload at stage $s$ is defined by

$$WL_s = \frac{1}{\bar{B}_s m_s} \sum_{j=1}^{n} p_{js} s_j. \tag{25}$$

The algorithm yields the most balanced set of numbers of machines at the first and second stage $(m_1^*, m_2^*)$, with the smallest absolute difference of workloads $|WL_1 - WL_2|$. From the most balanced situation, a bottleneck situation is achieved by shifting one machine to the opposite stage (cf. Tan et al., 2018, for details). Problem instances with a bottleneck at each stage and the situation

with the most balanced distribution of machines are considered in the experimentation. Release dates of the jobs are set according to

$$r_j \sim DU\left[0, \left\lceil \alpha \left( \frac{1}{\bar{B}_1 m_1^*} \sum_{j=1}^{n} s_j p_{f(j),1} + \frac{1}{\bar{B}_2 m_2^*} \sum_{j=1}^{n} s_j p_{f(j),2} \right) \right\rceil \right]. \tag{26}$$

The parameter $\alpha \in \{0.25,\ 0.75\}$ is multiplied with a crude estimate of the makespan to create tight and wide ranges of release dates. The parameter $m_s^*$ is equal to the number of machines at stage $s$ in the most balanced setting obtained by the BC algorithm. Due dates are set based on the release dates adding the job's overall processing time multiplied by a flow factor of $FF = 1.3$. The quantity $FF$ is an indicator for the expected average waiting time of the flow shop.

Another set of experiments with very small-sized problem instances is designed to compare the heuristics against a benchmark provided by solving the MILP formulation of Section 2.2. These instances feature two families at the first stage with five jobs each. A total number of three machines with a batch capacity of 10 pieces is used. The remaining values correspond to the ones of the regular instances. Ten instances per factor combination are randomly generated with a different seed. The design of experiments is summarized in Table 1.

### 5.2. Parameter setting and implementation issues

Any heuristic is allowed a given period of computing time to yield a solution. In the case of the IDA, it is therefore necessary to distribute that time between multiple executions of the procedure solving the subproblem. There is a trade-off between a higher number of iterations and more computing time allowed to solve each individual subproblem, likely facilitating better solutions thereof. It seems hard to determine a reasonable allocation of time valid for a larger set of problem instances without conducting extensive preliminary studies. Therefore, instead of prescribing a fixed number of iterations to be processed within the overall time limit, convergence is used as the stopping criterion for solving the subproblem (see below). The actual number of iterations is thus dependent on the time used for solving the subproblems.

Some preliminary experimentation and Tan et al. (2018) for a related p-batching problem show a significant influence of the choice of parameters on the solution quality of IDA approaches. The values are set to $\alpha = 0.66$ and $\beta = 1.66$. These value combinations proved to lead to the best results from the set, that is, parameter grid $\{0.0,\ 0.33,\ 0.66,\ 1.0,\ 1.33,\ 1.66,\ 2.0\} \times \{0.0,\ 0.33,\ 0.66,\ 1.0,\ 1.33,\ 1.66,\ 2.0\}$ in preliminary experiments with a limited number of problem instances.

To further assess this setting against the best choice from the combinations for each instance, additional experiments are conducted where the IDA-ILS is performed with each of the 49 possible parameter combinations from the grid. The experiments are limited to the 48 problem instances with 45 jobs and a bottleneck at either the first or the second stage. A computing time limit of 60 seconds per instance is prescribed, resulting in a computing time requirement of 49 minutes per instance for the grid search.

Table 1
Design of experiments

| Factor | Level | Count |
|---|---|---|
| Number of stages $S$ | 2 | 1 |
| Number of families $F_1$ for stage 1 | $2^*$, 3, | 1 |
| Number of families $F_2$ for stage 2 | Each first-stage family will become either one or two families with equal probability at the second stage. If a family becomes two families at the second stage, an operation is assigned to either one of these with a probability of 0.5 | 1 |
| Number of jobs per family $F_1$ | $5^*$, 10, 15, 20 | $1^*/3$ |
| Total number of machines | $3^*$, 6 | 1 |
| Bottleneck machine at stage | 1, 2, balanced | $2^*/3$ |
| Job sizes $s_j$ | $s_j \sim DU[1,\ 8]$ | 1 |
| Batch capacity $B_s$ | $10^*$, $\{15,\ 30\} \times \{15,\ 30\}$ | $1^*/4$ |
| Family processing times $p_{fs}$ | 2 with probability 0.2 | 1 |
| | 4 with probability 0.2 | |
| | 10 with probability 0.3 | |
| | 16 with probability 0.2 | |
| | 20 with probability 0.1 | |
| Job weight $w_j$ | $w_j \sim DU[1,\ 5]$ | 1 |
| Job release date $r_j$ | $r_j \sim DU[0,\ [\alpha(\frac{1}{B_1 m_1^*}\sum_{j=1}^{n} s_j p_{f(j),1} + \frac{1}{B_2 m_2^*}\sum_{j=1}^{n} s_j p_{f(j),2})]].\alpha \in \{0.25,\ 0.75\}$ | 2 |
| Job due dates $d_j$ | $d_j := r_j + FF \cdot s_j(p_{f(j),1} + p_{f(j),2})$ | 1 |
| Flow Factor FF | 1.3 | 1 |
| Setup time $q_s$ | $q_s := \lfloor \frac{s_j^{\max} - s_j^{\min}}{4} p_s^{\max} \rfloor$ | 1 |
| Instances per factor combination | | $10^*/3$ |
| Total | | $40^*/216$ |

*Small-sized instances

The same subset of problem instances is used for experiments to measure the performance of the ILS-DG against the ILS-DG-MS with a large amount of computing time. The computing time limit of the ILS-DG is set to 49 minutes. The ILS-DG-MS is configured to restart seven times with a computing time limit of seven minutes, resulting in a total computing time requirement of also 49 minutes per instance. The results produced by these two heuristics are also evaluated in the context of the results obtained by the above-described IDA-ILS grid search configuration.

Preliminary experiments are conducted with 20 randomly chosen instances from the set of large-sized instances described in Table 1 to find appropriate parameter values for the meta-heuristics. For the GGA, results from experiments with each combination of population size $n_{pop} \in \{150,\ 300,\ 450\}$ and mutation probability $p_m \in \{0.01,\ 0.05,\ 0.1\}$ are compared. Consequently, the parameters are set to $n_{pop} = 150$ and $p_m = 0.05$. Ten chromosomes of each generation are modified with LS. The crossover probability is set as $p_c = 0.9$, and the replacement rate is $p_r = 0.8$ after performing some preliminary experiments with $p_c \in \{0.80,\ 0.85,\ 0.90\}$ and $p_r \in \{0.75,\ 0.80,\ 0.85\}$. After five generations without improvement, the algorithm terminates and returns control to the IDA. Regarding the ILS, results of preliminary computational experiments with a number of perturbation steps from the set $\{2,\ 5,\ 10\}$ are compared. The best results are achieved with two perturbation steps, which are therefore adopted for all further experiments. After 10 iterations without improvement, the number of perturbation steps is increased to four to support escaping local optima. The heuristic terminates after 25 iterations without improvement. The probabilities for the different move types used in the perturbation phase of the proposed ILS schemes are determined based on a trial and error strategy. Whenever the ATC dispatching rule with index (18) is applied, the look-ahead parameter $\kappa$ is taken from the interval [0.1, 2.5] with a step size of 0.1 (cf. Tan et al., 2018). Each problem instance is solved five times with different seeds to obtain statistically significant results.

Factor values relevant only for the set of small-sized instances are indicated with an asterisk.

All algorithms are coded using the C++ programming language. The GGA implementation makes use of the GAlib framework (Wall, 2023). We implement the graph representation with the help of the boost framework (Boost, 2023) in version 1.72. The MILP formulation of Section 2.2 is implemented with the commercial solver IBM ILOG CPLEX 12.1. All experiments are conducted on an Intel Core i7-2600 CPU 3.40GHz computer with 16GB of RAM.

### 5.3. Computational results

#### 5.3.1. Results for small-sized instances

We first present the results of the experiments with the small-sized problem instances. Instead of presenting the objective function values, we report the heuristic ratio of an approach:

$$HR_A := OBJ_A / OBJ_*, \tag{27}$$

where $OBJ_A$ represents the average objective function value of an approach $A$, and $OBJ_*$ stands for the objective function value of a benchmark approach. CPLEX is used to provide benchmark solutions with a fixed computing time limit of 60 minutes per instance. A computing time limit of 30 seconds is granted for the applied heuristics. The average values from the 10 corresponding

Table 2
Results of small-sized problem instances

| Bottleneck stage | Range of releases | IDA-GGA | IDA-ILS | ILS-DG | CPLEX | Ø gap |
|---|---|---|---|---|---|---|
| 1 | Tight | 1.008 | 1.018 | 1.009 | **1.000** | 90.94% |
| | Wide | **0.995** | 1.005 | 1.039 | 1.000 | 92.45% |
| 2 | Tight | 1.039 | 1.048 | 1.045 | **1.000** | 11.08% |
| | Wide | 1.013 | 1.015 | 1.017 | **1.000** | 7.16% |
| Average | | 1.014 | 1.021 | 1.027 | **1.000** | |

instances are presented for each factor combination. The relative mixed integer programming (MIP) gap of a solution obtained by CPLEX is computed as

$$gap := \left|\text{best bound} - obj^*\right| / \left|obj^*\right|, \tag{28}$$

where the abbreviation $obj^*$ is used for the objective function value. Average values are reported. The best solutions per row are indicated by a bold typeface. The obtained results are shown in Table 2.

Instances with a bottleneck at the first stage appear harder to solve as for these CPLEX cannot find results with a proven optimality, and the average relative MIP gap is larger than 90%. The range of the release dates does not have a significant influence on the hardness. The average objective function values obtained by the heuristics are close to the ones of the benchmark solutions and in the case of a wide distribution and a bottleneck in stage 1 even slightly below. IDA-GGA can find the overall best results within 1% of the benchmark solutions. Optimality of the benchmark solutions is proven for eight out of the 10 instances with the bottleneck in stage 2 and tight or wide ranges of release dates. All approaches can reliably find the optimal solutions for three to four of the instances with tight and five to six instances of the instances with a wide range of release dates in all five executions of the heuristic with a different seed. On average, the objective function value deviations are within 2% to 5% of the solutions found with CPLEX. The results demonstrate that the implementations of the algorithms are correct, and the heuristics are, in principle, able to find optimal or near-to-optimal schedules.

### *5.3.2. Results for large-sized instances*

Next, the results of experiments with large-sized instances are examined. As CPLEX cannot be used as a benchmark, only solutions from the heuristic approaches are compared. A computing time limit of 360 seconds is applied. Again, average TWT values from five independent replications of the heuristics are considered. A Wilcoxon signed-rank test (Wilcoxon, 1945) with a significance level of 1% reveals statistically significant differences between the examined results.

The IDA-ILS scheme yields the overall best results and is therefore declared the reference with $HR_{IDA-ILS} = 1.0$. Average values across groups of instances with similar factor values are computed. As no significant influence of the batch capacity at the first or second stage on the relative performance could be observed, this factor is not displayed in detail. We rather aggregate groups of

Table 3
Results for large-sized problem instances

| Number of jobs | Bottleneck stage | Range of releases | IDA-GGA | IDA-ILS | ILS-DG |
|---|---|---|---|---|---|
| 30 | 1 | Tight | 1.10 | **1.00** | 1.06 |
| | | Wide | 1.13 | **1.00** | 1.11 |
| | 2 | Tight | 1.14 | **1.00** | 1.04 |
| | | Wide | 1.18 | **1.00** | 1.06 |
| | Balanced | Tight | 1.15 | **1.00** | 1.07 |
| | | Wide | 1.21 | **1.00** | 1.12 |
| 45 | 1 | Tight | 1.24 | **1.00** | 1.07 |
| | | Wide | 1.30 | **1.00** | 1.11 |
| | 2 | Tight | 1.20 | **1.00** | 1.06 |
| | | Wide | 1.26 | **1.00** | 1.08 |
| | Balanced | Tight | 1.27 | **1.00** | 1.10 |
| | | Wide | 1.37 | **1.00** | 1.17 |
| 60 | 1 | Tight | 1.35 | **1.00** | 1.08 |
| | | Wide | 1.45 | **1.00** | 1.13 |
| | 2 | Tight | 1.27 | **1.00** | 1.08 |
| | | Wide | 1.35 | **1.00** | 1.10 |
| | Balanced | Tight | 1.36 | **1.00** | 1.11 |
| | | Wide | 1.45 | **1.00** | 1.17 |
| Average | | | 1.27 | **1.00** | 1.10 |

instances with the same number of jobs, the same range of ready times, and the same BC to focus on factors of impact. Consequently, each line shown in Table 3 corresponds to the average value of 12 individual instances.

The data indicate a clear hierarchy with the IDA-ILS at the top and the IDA-GGA at the bottom regarding the quality of solutions. The IDA-ILS scheme outperforms the ILS-DG by between 4% and 17%. The advantage grows only slightly with an average of around 1% to 5% with increasing number of jobs from 30 to 60. A similar difference can be observed in relation to the bottleneck stage, where with a bottleneck at the first stage, the advantage of the IDA-ILS is around 2%–4% higher than if the bottleneck is at the second stage. For the balanced situation, an advantage of 3%–9% can be observed. Instances with a balanced distribution of machines and a wide range of ready times appear to be the hardest for the ILS-DG. The TWT values are up to 17% worse on average, compared to the IDA-ILS. Obviously, the IDA-GGA is no match for the other two approaches in the face of larger problem instances. The quality of solutions deteriorates by up to 45%, compared to the benchmark.

A much stronger sensitivity to the number of jobs can be observed, compared to the other approaches. A reason may be a computational overhead caused by the population-based GGA. More lightweight approaches such as ILS may utilize the given computing time more efficiently and therefore appear more appropriate. We can also again observe the largest differences for instances with a wider range of release dates and a bottleneck at the first stage.

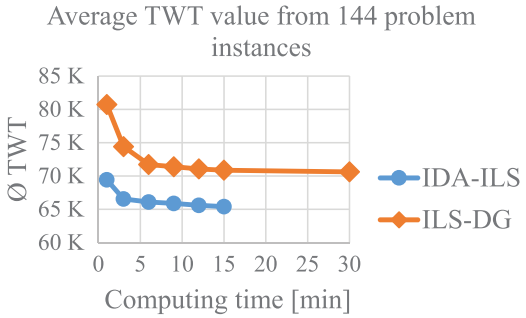



Fig. 7. Influence of the computing time limit on the solution quality.

Table 4
Results from the IDA-ILS procedure with different parameter settings

| Number of jobs | Bottleneck stage | Range of releases | $(\alpha, \beta)$ grid | $\alpha = 0.66$, $\beta = 1.66$ | Rel. difference |
|---|---|---|---|---|---|
| 45 | 1 | Tight | 86,469 | 87,708 | −1.4% |
| | | Wide | 73,757 | 74,949 | −1.6% |
| | 2 | Tight | 66,673 | 71,246 | −6.4% |
| | | Wide | 56,437 | 60,357 | −6.5% |
| Average | | | 70,834 | 73,565 | −3.7% |

### *5.3.3. Dependency of the solution quality on the computing time*

Further experiments are conducted with the problem instances with a bottleneck at either the first or the second stage and varying time limits to further investigate the gap between the IDA-ILS and the ILS-DG. The IDA-ILS is tested with a computing time limit of 1, 3, 6, 9, and 15 minutes. The ILS-DG is in addition tested with a 30-minute computing time limit per instance. The average TWT values across all 144 problem instances in relation to the computing time limit are displayed in Fig. 7.

One can see that already with a computing time limit of one minute, the IDA-ILS can provide fairly high-quality schedules. The decrease of TWT is steep with the increase up to three minutes, and after that, the slope remains rather flat. For the ILS-DG, a steeper decrease is observed for a computing time limit of up to six minutes. On average, even 30 minutes of computing time is not sufficient to achieve results on the level of the former approach obtained within a single minute, even though for subsets of instances, especially with a higher number of jobs, at least this can be observed. The superiority of the IDA-ILS over the competing approaches presented in this paper, however, can be clearly seen.

### *5.3.4. Dependency of the solution quality on the IDA parameterization*

The difference of the average TWT values obtained by applying the IDA-ILS with the best-performing values of $\alpha$ and $\beta$ taken from the grid, compared to the ones obtained by a single run of 60 seconds with $\alpha = 0.66$ and $\beta = 1.66$, are presented in Table 4. Again, the Wilcoxon

Table 5
Results from experiments with extended computing time limit

| Number of jobs | Bottleneck stage | Range of releases | IDA-ILS ($\alpha$, $\beta$) grid | ILS-DG | ILS-DG-MS |
|---|---|---|---|---|---|
| 45 | 1 | Tight | **1.000** | 1.081 | 1.074 |
| | | Wide | **1.000** | 1.145 | 1.130 |
| | 2 | Tight | **1.000** | 1.064 | 1.085 |
| | | Wide | **1.000** | 1.091 | 1.119 |
| Average | | | **1.000** | 1.095 | 1.102 |

signed-rank test (Wilcoxon, 1945) with a significance level of 1% confirms statistically significant differences between the results obtained by the different approaches.

On average the extensive trial of parameter values leads to TWT values, which are around 1.5% lower with a bottleneck at the first stage and around 6.5% with a bottleneck at the second stage. However, due to the significantly higher computing time requirements, the comparison is only of theoretical interest. But the observed results indicate that the initial setting of the parameter values is in principle suited to find high-quality schedules. Further improving the performance by a more tailored choice of parameters for instances with different characteristics may be promising.

The objective function values found using the IDA-ILS with the grid configuration are compared to the ones obtained using the ILS-DG and the ILS-DG-MS, also achieved with a given overall computing time limit of 49 minutes. The heuristic ratios are shown in Table 5.

Again, the IDA-ILS is used as a benchmark. This is quite remarkable as the solutions are actually obtained within 60 seconds of computing time per instance, although profiting from a strongly tuned set of $\alpha$ and $\beta$ parameters. Even with 49 minutes of computing time per instance, the TWT values found using the ILS-DG are around 10% higher. The results provided by the ILS-DG only slightly outperform the ones found by the ILS-DG-MS approach. The Wilcoxon signed-rank test (Wilcoxon, 1945) with a significance level of 1% confirms statistically significant differences between the different results.

## 6. Conclusion and future research

In this paper, we discussed a two-stage flexible flow-shop scheduling problem with s-batching characteristics. A maximum batch size was assumed for each machine at any of the two stages. Moreover, only jobs belonging to the same family can be batched together. A constant setup time occurs between the processing of consecutive batches. We proposed a stage-based decomposition scheme that iteratively computes appropriate due dates for the jobs at the first stage. The resulting subproblems for parallel machines were solved by a GGA and by an ILS scheme. Moreover, an alternative ILS scheme was proposed that was based on a disjunctive graph formulation of the scheduling problem at hand. By designed computational experiments, we demonstrated that the decomposition scheme based on the ILS scheme outperforms the two remaining approaches. High-quality solutions were computed using a reasonable amount of computing time.

There are several directions for future research. First of all, the two-stage flexible flow shop can be replaced by a $l$-stage flexible flow shop where $l > 2$. The different stages can contain machines with s-batching or p-batching characteristics. Moreover, it is interesting to assume reentrant flows in the flow shop, that is, the same stage is visited several times by the same stage. Such process conditions appear in semiconductor wafer fabrication facilities (wafer fabs) (cf. Mönch et al., 2013). Job shops might be considered instead of flow shops. It seems also worthwhile to study parallelization schemes for the different subproblems that appear within the decomposition approach.

## Acknowledgments

None

## References

Bilyk, A., Mönch, L., Almeder, C., 2014. Scheduling jobs with ready times and precedence constraints on parallel batch machines using metaheuristics. *Computers & Industrial Engineering* 78, 175–185.

Boost., 2023. Boost C++ libraries. Available at https://www.boost.org/ (accessed 6 March 2023).

Cheng, T.C.E., Kovalyov, M.Y., 2001. Single machine batch scheduling with sequential job processing. *IIE Transactions* 33, 5, 413–420.

Chrétienne, P., Hazır, Ö., Kedad-Sidhoum, S., 2011. Integrated batch sizing and scheduling on a single machine. *Journal of Scheduling* 14, 541–555.

Dauzère-Pérès, S., Paulli, J., 1997. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research* 70, 281–306.

Demirkol, E., Uzsoy, R., 2000. Decomposition methods for reentrant flow shops with sequence-dependent setup times. *Journal of Scheduling* 3, 155–177.

Emmons, H., Vairaktarakis, G., 2013. *Flow Shop Scheduling: Theoretical Results, Algorithms, and Applications*. Springer, New York.

Falkenauer, E., 1998. *Genetic Algorithms and Grouping Problems*. Wiley, Chichester.

Falkenauer, E., 1996. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics* 2, 1, 5–30.

Fowler, J.W., Mönch, L., 2022. A survey of scheduling with parallel batch (p-batch) processing. *European Journal of Operational Research* 298, 1, 1–24.

Gahm, C., Wahl, S., Tuma, A., 2022. Scheduling parallel serial-batch processing machines with incompatible job families, sequence-dependent setup times and arbitrary sizes. *International Journal of Production Research* 60, 17, 5131–5154.

Gibson, I., Rosen, D., Stucker, B., Khorasani, M., 2021. *Additive Manufacturing Technologies*. Springer, Cham.

Gleeson, D., Jakobsson, S., Salman, R., Ekstedt, F., Sandgren, N., Edelvik, F., Carlson, J.S., Lennartson, B., 2022. Generating optimized trajectories for robotic spray painting. *IEEE Transactions on Automation Science and Engineering* 19, 3, 1380–1391.

Goldberg, D.E., 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* 5, 287–326.

Hansen, P., Mladenovic, N., 2001. Variable neighborhood search: principles and applications. *European Journal of Operational Research* 130, 449–467.

Hansen, P., Mladenovic, N., Moreno Pérez, J.A., 2010. Variable neighbourhood search: methods and applications. *Annals of Operations Research* 175, 1, 367–407.

He, C., Lin, H., Lin, Y., 2015. Bounded serial-batching scheduling for minimizing maximum lateness and makespan. *Discrete Optimization* 16, 70–75.

Koulamas, C. 1998. A guaranteed accuracy shifting bottleneck algorithm for the two-machine flowshop total tardiness problem. *Computers & Operations Research* 25, 2, 83–89.

Knopp, S., Dauzère-Pérès, S., Yugma, C., 2017. A batch-oblivious approach for complex job-shop scheduling problems. *European Journal of Operational Research* 263, 1, 50–61.

Lourenço, H.R., Martin O., Stützle T., 2010. Iterated local search: framework and applications. In Gendreau, M., Potvin, J.-V. (eds), *Handbook of Metaheuristics* (2nd edn.). Kluwer Academic Publishers, Dordrecht, pp. 363–397.

Michalewicz, Z., 1996. *Genetic Algorithms + Data Structures = Evolution Programs* (3rd edn.). Springer, Berlin.

Mönch, L., Fowler, J.W., Mason, S.J., 2013. *Production Planning and Control for Semiconductor Wafer Fabrication Facilities: Modeling, Analysis, and Systems*. Springer, New York.

Mosheiov, G., Oron, D., 2008. A single machine batch scheduling problem with bounded batch size. *European Journal of Operational Research* 187, 1069–1079.

Nsengimana, J., van der Walt, J., Pei, E., Miah, M., 2019. Effect of post-processing on the dimensional accuracy of small plastic additive manufactured parts. *Rapid Prototyping Journal* 25, 1, 1–12.

Ovacik, I.M., Uzsoy, R., 1997. *Decomposition Methods for Complex Factory Scheduling Problems*. Springer, Boston.

Panwalkar, S.S., Koulamas, C., 2019. The Evolution of schematic representations of flow shop scheduling problems. *Journal of Scheduling* 22, 379–391.

Paulli, J., 1995. A hierarchical approach for the FMS scheduling problem. *European Journal of Operational Research* 86, 32–42.

Potts, C.N., Kovalyov, M.Y., 2000. Scheduling with batching: a review. *European Journal of Operational Research* 120, 2, 228–249.

Queiroga, E., Pinheiro, R.G.S., Christ, Q., Subramanian, A., Pessoa, A.A., 2021. Iterated local search for single machine total weighted tardiness batch scheduling. *Journal of Heuristics* 27, 3, 353–438.

Rocholl, J., Mönch, L., Fowler, J., 2020. Bi-criteria parallel batch machine scheduling to minimize total weighted tardiness and electricity cost. *Journal of Business Economics* 90, 1345–1381

Rocholl, J., Mönch, L., 2023. Scheduling jobs in flexible flow shops with s-batching machines using metaheuristics. Proceedings of the 14th International Conference on Metaheuristics (MIC 2022), LNCS 13838, July 11–14, Syracuse, Italy, pp. 560–568.

Shen, L., Mönch, L., Buscher, U., 2014. Simultaneous and iterative approach for parallel machine scheduling with sequence dependent family setups. *Journal of Scheduling* 17, 5, 471–487.

Sobeyko, O., Mönch, L., 2011. A comparison of heuristics to solve a single machine batching problem with unequal ready times of the jobs. Proceedings of the 2011 Winter Simulation Conference, December 11–14, Phoenix, AZ, pp. 2011–2020.

Sobeyko, O., Mönch, L., 2015. Grouping genetic algorithms for solving single machine multiple orders per job scheduling problems. *Annals of Operations Research* 235, 1, 709–739.

Stützle, T., Ruiz, R., 2018. Iterated local search. *Handbook of Heuristics*. Springer, Cham.

Suppiah, S., Omar, M.K., 2014. A hybrid tabu search for batching and sequencing decisions in a single machine environment. *Computers & Industrial Engineering* 78, 135–147.

Tan, Y., Mönch, L., Fowler, J., 2018. A hybrid scheduling approach for a two-stage flexible flow shop with batch processing machines. *Journal of Scheduling* 21, 2, 209–226.

Uzunoglu, A., Gahm, C., Wahl, S., Tuma, A., 2023. Learning-augmented heuristics for scheduling parallel serial-batch processing machines. *Computers & Operations Research* 151, 106122.

Vepsalainen, A., Morton, T.E., 1987. Priority rules and lead time estimates for job shop scheduling with weighted tardiness costs. *Management Science* 33, 1036–1047.

Voß, S., Woodruff, D.L., 2006. *Introduction to Computational Optimization Models for Production Planning in a Supply Chain* (2nd edn.). Springer, New York.

Wall, M., 2023. Galib. A C++ library of genetic algorithms components. Available at http://lancet.mit.edu/ga/ (accessed 6 March 2023).

Webster, S., Baker, K., 1995. Scheduling Groups of jobs on a single machine. *Operations Research* 43, 4, 692–703.

Wilcoxon, F., 1945. Individual comparisons by ranking methods. *Biometrics* 1, 80–83.

Yang, Y., Kreipl, S., Pinedo, M., 2000. Heuristics for minimizing total weighted tardiness in flexible job shops. *Journal of Scheduling* 3, 89–108.

## Appendix A: Computational Example

Additional data for the example shown in Fig. 1 is provided in Table A1. The table also contains the completion times of the jobs on each stage, the processing times of the batch job $j$ belongs to on each stage $s$ (denoted by $b_{js}$), and the resulting weighted tardiness values $w_j T_j$ for each job $j$. The family of job $j$ on stage $s$ is $f_{js}$. The TWT value of the schedule is also reported in the table.

The schedule is visualized by means of a Gantt chart in Fig. A1. The shaded areas indicate setups on the machines due to processing of consecutive batches.

Table A1
Problem instance data

| Job | $s_j$ | $f_{j1}$ | $r_j$ | $d_j$ | $w_j$ | $b_{j1}$ | $C_{j1}$ | $f_{j2}$ | $b_{j2}$ | $C_{j2}$ | $w_j T_j$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 7 | 26 | 3 | 4 | 21 | 1 | 6 | 27 | 3 |
| 2 | 1 | 2 | 6 | 23 | 1 | 9 | 15 | 1 | 6 | 27 | 4 |
| 3 | 3 | 3 | 3 | 23 | 2 | 6 | 18 | 1 | 8 | 32 | 18 |
| 4 | 2 | 1 | 4 | 21 | 1 | 6 | 10 | 2 | 6 | 21 | 0 |
| 5 | 1 | 1 | 3 | 14 | 3 | 6 | 10 | 1 | 8 | 32 | 54 |
| 6 | 1 | 3 | 4 | 17 | 1 | 6 | 15 | 3 | 1 | 36 | 19 |
| 7 | 1 | 2 | 6 | 21 | 1 | 9 | 15 | 2 | 6 | 21 | 0 |
| 8 | 1 | 2 | 4 | 17 | 3 | 9 | 15 | 3 | 3 | 18 | 3 |
| 9 | 2 | 3 | 3 | 19 | 1 | 4 | 7 | 3 | 3 | 18 | 0 |
| 10 | 2 | 3 | 7 | 19 | 3 | 6 | 15 | 2 | 4 | 34 | 45 |
| | | | | | | | | | | **TWT** | **146** |

| **Setup time per stage** | | **Unit processing times by family and stage** | | | | | |
|---|---|---|---|---|---|---|---|
| $q_1$ | $q_2$ | $p_{11}$ | $p_{21}$ | $p_{31}$ | $p_{12}$ | $p_{22}$ | $p_{32}$ |
| 2 | 3 | 2 | 3 | 2 | 2 | 2 | 1 |

## Appendix B: RVND

The pseudocode of the RVND algorithm is show next.

**Algorithm RVND(input**: initial schedule $S$, set of neigborhood structures $N$)

```
randomly sort the set of neighborhood structures N
k ← 1
while k ≤ |N|
   S′ ← bestFit(N_k)
   if TWT(S′) < TWT(S)
      S ← S′, k ← 1
      continue at Step 1
   else
      k ← k + 1
return S
```

Here, the bestFit procedure explores the neighborhood $N_k(S)$ and provides a schedule $S'$ with smallest TWT value from the solutions that belong to the neighborhood.

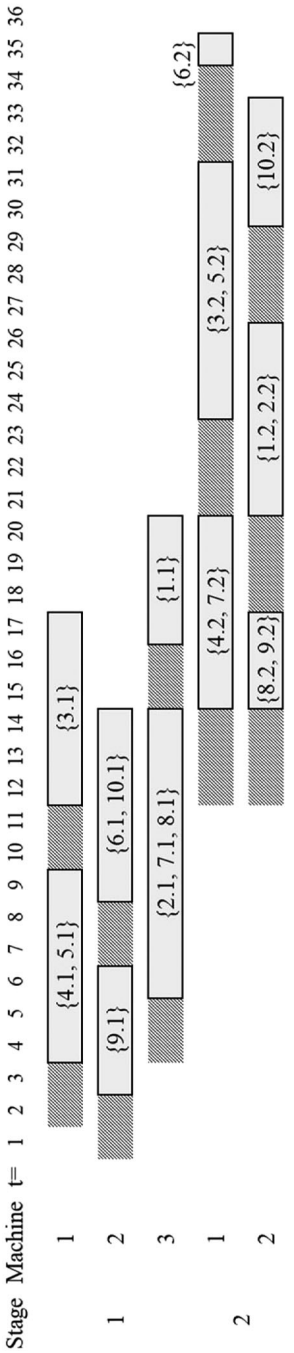



Fig. A1. Gantt chart of the schedule from Fig. 1.

## Appendix C: ILS-DG-MS

Next, the pseudocode of the ILS-DG-MS procedure is summarized.

**Algorithm ILS-DG-MS(input**: termination criteria $t1$, $t2$)

```
1  while(not t1)
2    initialize solution S
3    While(not t2)
4      // Local search
5      S ← RVND(S),
6      S* ← S
7      // perturbation
8      S ← PERTURB(S)
9  return S*
```

Here, the RVND procedure with different neighborhood structures is taken from Appendix B, whereas the PERTURB procedure is described in Section 4.3.2. The termination criteria $t1$, $t2$ are typical maximum allowed computing times. In the case of the ILS-DG variant, only a single pass is carried out starting from Step 1.