Proceedings of the 2025 Winter Simulation Conference
E. Azar, A. Djanatliev, A. Harper, C. Kogler, V. Ramamohan, A. Anagnostou, and S. J. E. Taylor, eds.

SOLVING FLEXIBLE FLOW-SHOP SCHEDULING PROBLEMS WITH MAXIMAL TIME LAGS: A COMPUTATIONAL STUDY

Daniel Schorn[1] and Lars Mönch[1]

[1]Dept. of Mathematics and Computer Science, University of Hagen, Hagen, GERMANY

ABSTRACT

We consider a scheduling problem for a two-stage flexible flow shop with maximal time lags between consecutive operations motivated by semiconductor manufacturing. The jobs have unequal ready times and both initial and inter-stage time lags. The total weighted tardiness is the performance measure of interest. A heuristic scheduling framework using genetic programming (GP) to automatically discover priority indices is applied to the scheduling problem at hand. Computational experiments are carried out on randomly generated problem instances. The results are compared with the ones of a reference heuristic based on a biased random-key genetic algorithm combined with a backtracking procedure and a mixed-integer linear programming-based decomposition approach. The results show that high-quality schedules are obtained in a short amount of computing time using the GP approach.

1 INTRODUCTION

Dispatching and scheduling is a crucial production control function in most semiconductor wafer fabrication facilities (wafer fabs). Production control for wafer fabs is complicated due to the processing conditions and the sheer size of the wafer fabs in terms of number of machines, products, and operations. The moving objects in these facilities are jobs that consist of up to 25 wafers, thin discs often made of silicon or gallium arsenide. Maximal time lags are an important process restriction in wafer fabs (Han and Lee 2024; Schorn and Mönch 2024). Maximal time lags are installed by process engineers to prevent native oxidation and contamination effects on the wafer surface (Scholl and Domaschke 2000). In the case of maximal time lag violations, it is likely that the affected jobs will be scrapped. Maximal time lags in wafer fabs can be nested, i.e. overlapping. The appropriate treatment of maximal time lags in production control approaches for wafer fabs is still not very well understood (May et al. 2024). It is well known that conventional dispatching rules that are myopic by nature have difficulties to deal with time lags. In this paper, we will demonstrate that this is not the situation for automatically discovered dispatching rules using GP (Branke et al. 2016) which are able to learn to avoid maximal time lag violations. We will demonstrate that such advanced dispatching strategies are able to determine schedules using a short amount of computing time (CT) that can outperform those provided by metaheuristic approaches hybridized with time-consuming mixed-integer linear programming (MILP)- or constraint programming (CP)-based decomposition approaches (Klemmt and Mönch 2012). GP approaches are only applied so far to oversimplified scheduling situations with time lags (Quin et al. 2021). In the present paper, we still apply our approach only to a simplified model problem which is a two-stage flexible flow shop. However, in contrast to previous work the maximal time lags might have different lengths and initial maximal time lags are also taken into account.

The main purpose of the present paper is to compare the performance of the heuristic scheduling framework proposed by Schorn and Mönch (2025) for the special case of non-batching machines and the total weighted tardiness measure with a job-based decomposition approach which extends a scheduling approach proposed by Klemmt and Mönch (2012) for a slightly different problem. Such a comparison is

desirable since reference approaches for flexible flow shops or even job shops with maximal time lags are still rare.

The paper is organized as follows. In the next section, we describe the scheduling problem at hand, discuss related work, and derive research questions. The different solution approaches including the GP approach are discussed in Section 3. Results of computational experiments are reported in Section 4. Finally, conclusions and future research directions are provided in Section 5.

2 PROBLEM SETTING AND ANALYSIS

2.1 Scheduling Problem

We consider a two-stage flexible flow shop with m_s identical parallel machines on each stage $s \in \{1,2\}$. Jobs $j = 1, \ldots, n$ must be scheduled. Each job belongs to a family $f = 1, \ldots, F$, where all jobs of a family have the same processing time p_{sj} for their operation executed on stage s. Furthermore, each job j has a ready time $r_j \geq 0$, a due date d_j, and a weight w_j which is used to express the importance of job j. The completion time of job j is C_j. The total weighted tardiness (TWT) measure is given by $\sum_{j=1}^{n} w_j T_j$ for the tardiness $T_j \coloneqq \left(C_j - d_j\right)^+$ of job j. Here, we abbreviate $x^+ \coloneqq \max(x, 0)$. The start time of the operation of job j on stage s is denoted by S_{sj}. A maximal time lag of length t_{12j} between the operations executed on both stages is ensured for job j if we have $S_{2j} \leq S_{1j} + t_{12j}$. Initial time lags are only possible for the first stage. They are satisfied if $S_{1j} \leq t_{1j}$ holds. Using the three-field notation for deterministic machine scheduling, the scheduling problem is represented by

$$FF2\left|r_j, t_{1j}, t_{12j}\right|TWT, \tag{1}$$

where $FF2$ and r_j refer to a two-stage flexible flow shop and unequal ready times, respectively. Initial and inter-stage, i.e. regular time lags are also indicated. Problem (1) is NP-hard since the special case $F2||TWT$ with no time lags already has this property. Hence, we have to look for efficient heuristics to tackle large-sized instances of (1) using an appropriate amount of CT.

2.2 Discussion of Related Work and Research Questions

Dealing with maximal time lags in wafer fabs is challenging (Mönch et al. 2011). Rule-based approaches and deterministic scheduling approaches are established methods. The rule-based approaches are formed by dispatching strategies combined with stopping strategies to avoid time lag violations (cf. Scholl and Domaschke 2000; Zhang et al. 2016; Kopp et al. 2020 among others). A stopping strategy is similar to an order release approach, i.e., a job is set on hold if it is likely that a time lag will be violated if the job is processed next. It is well-known that for nested time lags dispatching is not appropriate since dispatching rules are myopic. Scheduling approaches are either MILP- or CP-based in combination with job-based decomposition approaches by solving a series of smaller scheduling problems or based on metaheuristics. Decomposition heuristics and metaheuristics can be hybridized. Examples of the first class are Klemmt and Mönch (2012), Jung et al. (2014), and Cailloux and Mönch (2019). Metaheuristics based on neighborhood search that accept moves only if time lags are not violated belong to the second class of approaches (cf. Yugma et al. 2012; Knopp 2016; Han and Lee 2023). Simulated annealing, greedy randomized adaptive search, and genetic algorithms (GAs) are applied in these papers to solve flow-shop and job-shop scheduling problems with maximal time lags. On the one hand, rules are more reactive and faster than deterministic scheduling approaches based on MILP, CP, or metaheuristics. On the other hand, designing the dispatching and stopping rules is manually carried out and time-consuming since it must be supported by discrete-event simulation. Consequently, a more automated discovery of such rules is desirable for production control problems with maximal time lags. GP is popular to automatically discover dispatching

rules (cf. Branke et al. 2016; Zhang et al. 2024 for related surveys). Complex job shops motivated by wafer fabs are tackled using GP and discrete-event simulation by Hildebrandt et al. (2014) and Kück et al. (2017). GP approaches are proposed by Braune et al. (2022) and Ferreira et al. (2022) to discover dispatching rules for flexible job shops. However, maximal time lags are not taken into account in these papers.

The most pertinent work for the present paper is the GP approach proposed by the two present authors in Schorn and Mönch (2025) for a two-stage flexible flow shop with batch processing machines and a blended performance measure including energy costs and TWT. A batch is a set of jobs that can be processed at the same time on a batch processing machine. In the present paper, we are interested in answering the following two research questions:

1. **RQ1:** Is it possible to learn dispatching rules that are able to avoid violations of maximal time lags as much as possible and are competitive with deterministic machine scheduling approaches?
2. **RQ2:** How robust are the learned rules to changes in structural properties of the problem instances used for training purposes when applied to instances with different properties?

RQ1 is an interesting question given the fact that especially manually designed dispatching rules are myopic by nature, whereas RQ2 is crucial taking into account the large computational burden of GP approaches due to the learning.

3 SOLUTION APPROACHES

3.1 Job-based Decomposition Heuristic

We extend the approach proposed by Klemmt and Mönch (2012) for an arbitrary flexible flow shop and total tardiness (TT) to the present situation of a two-stage flow shop with TWT measure. The original list scheduling approach works as follows:

1. Sort the n jobs in non-decreasing manner according to the earliest due date (EDD) rule.
2. Use list scheduling to insert the jobs in the order derived in Step 1 into the schedule. If maximal time lags are violated, use backtracking to repair the schedule.

A second approach is designed by solving a series of MILP instances. We always take the first $\tilde{n} < n$ jobs from the sorted list obtained from Step 1 into account. The MILP formulation for the scheduling problem is described in Klemmt and Mönch (2012). This is repeated until all jobs are scheduled. In each new MILP instance, the availability of the machines is updated, taking the scheduling decisions from the previous iteration into account.

First, we replace the EDD rule in the approach of Klemmt and Mönch (2012) by the following global Apparent Tardiness Cost (ATC) dispatching rule as proposed by Mönch and Zimmermann (2004), since it is well known that this rule often leads to small TWT values. The index value I_j of a job j at time t is given by:

$$I_j(t) := w_j/p_{1j}\, e^{-\left(d_j - p_{1j} + (r_j - t) - (\omega_{2j} + p_{2j})\right)^+ / \kappa \bar{p}}, \tag{2}$$

where κ is a look-ahead parameter that scales the slack and $\bar{p}$ is the average processing time of the not yet scheduled jobs on the first stage, and ω_{2j} is a waiting time estimate of job j on the second stage calculated iteratively by executing the schedule in a deterministic forward manner. For the first iteration, we initialize $\omega_{2j}^{(1)} := p_{2j}$, where the superscript (l) indicates the current iteration. After calculating the expected waiting times, the index values of all jobs are computed according to (2). In the following iterations, the waiting times are updated by $\omega_{2j}^{(l)} := (1-\gamma) \cdot \omega_{2j}^{(l-1)} + \gamma \cdot q_{2j}$, where q_{2j}, computed in the previous iteration,

represents the current waiting time for the second stage after executing the schedule. The current waiting time is the time elapsed between the completion of the job on the first stage and its starting time on the second stage. The smoothing factor $\gamma \in [0,1]$ is used to weigh the waiting time of the previous iteration and the current waiting time. This procedure terminates when a pre-defined number of iterations is reached or when the waiting time between two consecutive iterations is less than a prescribed threshold. If a regular time lag is violated, a backtracking procedure is applied that iteratively increases the starting time of the corresponding job on the first stage until the time lag is not violated anymore. Note that due to machine availabilities resulting from previously scheduled jobs, this can also delay the start time on the second stage.

The list scheduling approach is further improved by proposing job sequences, i.e. permutations, using a biased random-key GA (BRKGA) (Gonçalves and Resende 2011). We encode the n jobs by an array $(rk_1, \ldots, rk_n)$ where job j is represented by a random key $rk_j \in (0,1)$. Job s is before job t in the permutation if $rk_s \leq rk_t$ holds. A population of random-key vectors, called chromosomes, is considered. It is divided into a non-elite set and an elite set, which contains the best-performing chromosomes. A parameterized uniform crossover is applied where one of the parent chromosomes belongs to the elite set. Immigration is used to diversify the population. The list scheduling technique with backtracking from Klemmt and Mönch (2012) is used to decode each job sequence into a feasible schedule to evaluate the fitness of the chromosome by its TWT value. Finally, the best schedule obtained by the BRKGA is improved using the job sequence that leads to this schedule and applying the MILP-based decomposition heuristic from Klemmt and Mönch (2012) where we only replace the TT objective function by the one for TWT in the MILP formulation.

3.2 Genetic Programming Approach

We apply the heuristic scheduling framework of Schorn and Mönch (2025) to the special case of non-batching machines on each stage. It consists of four main procedures, some of them are partially parameterized using GP. The procedures are

1. iterative decomposition approach (IDA)
2. time window decomposition (TWD) approach
3. stopping strategy (STS)
4. backtracking procedure (BTP).

We first briefly summarize the four procedures belonging to the framework. The purpose of the IDA proposed by Tan et al. (2018) is to compute internal due dates for the operations to be scheduled on the first stage and ready times for the operations on the second stage. The IDA procedure is based on the parameters $\alpha, \beta \in [0,1]$ that weigh the ready times and the waiting times of the operations resulting from the schedule obtained in the previous iteration. Overall, l_{max} iterations are performed. Due to space limitations, we avoid describing the details and refer to Tan et al. (2018).

The outcome of the IDA allows to decompose the flow-shop scheduling problem into two subproblems for parallel identical machines. To compute the schedules for each stage within the IDA procedure, we use a list scheduling scheme coupled with the TWD approach similar to that proposed for batch scheduling by Mönch et al. (2005). However, since each machine can only process one job at a time in the present situation, the TWD approach basically ensures that only jobs are considered that are either ready for processing or arrive within a certain time window whose length is a fairly small multiple of the average processing time of the operations. A given dispatching rule is used to select the job from the time window to be processed next. This procedure is repeated whenever a machine or a new operation becomes available. For more details, we refer to Schorn and Mönch (2024). We use $\Delta_{s,max}$ as a parameter that sets the maximum length of the time window starting from time t for stage s. On contrast to the original TWD version, we consider all time windows with $0 \leq \Delta \leq \Delta_{s,max}$ at each decision point. The job with the highest

priority index j^* and the corresponding Δ value are chosen. Note that this allows for overlapping of time windows.

The TWD procedure is applied for both stages. However, when scheduling the operations for the first stage, a STS that controls the scheduling of jobs is applied before job j^* is added to the schedule. The STS ensures that no more than δ jobs without initial time lags become available to be processed on the second stage within a short time span, making it easier to satisfy regular time lags. For a machine k that becomes available at time t, the proposed STS procedure works as follows:

1. Introduce a list C_{est2} to store the estimated completion times of jobs on the second stage. It is initialized with $C_{est2} := \emptyset$, and $\tilde{C}_{2j}$ is the estimated completion time of job j on the second stage.
2. When a machine of the first stage becomes available at time t, we update C_{est2} by setting $C_{est2} := C_{est2} \backslash \{\tilde{C}_{2j} | \tilde{C}_{2j} \leq t\}$ because we expect that the corresponding jobs have finished processing on the second stage by time t.
3. Next, we check whether j^* is subject to an initial time lag or not. If either this is the case or if $|C_{est2}| < \delta$ holds, j^* is schedulable.
4. If j^* is schedulable, the completion time on the second stage is estimated by $\tilde{C}_{2j^*} := \begin{cases} S_{1j^*} + p_{2j^*} + t_{12j^*}, & \text{if } t_{12j^*} < \infty \\ S_{1j^*} + \widehat{FF} \cdot p_{2j^*}, & \text{otherwise.} \end{cases}$
   Here, $\widehat{FF}$ denotes an estimate of the flow factor (Mönch et al. 2013). If j^* has no initial time lag, we update $C_{est2} := C_{est2} \cup \tilde{C}_{2j^*}$. Finally, j^* is scheduled to start processing on machine k at time S_{j^*}.
5. If j^* is not schedulable, the procedure terminates, the availability time of machine k is increased by one and the TWD procedure is performed again.

Note that to prevent an increase in the number of initial time lag violations due to the use of the proposed STS, jobs with initial time lags do not affect C_{est2}. When scheduling the operations of the jobs on the second stage, a BTP is applied if the selected start time of job j^* leads to a violation of its regular time lag, i.e., if $S_{2j^*} > t_{12j^*} + S_{1j^*}$ holds. The main idea consists in delaying the start of the first operation of j^* on the first stage by setting $S_{1j^*} := S_{2j^*} - t_{12j^*}$. Due to space limitations, we refer to Klemmt and Mönch (2012) for details. The interaction of the four procedures of the framework is shown in Figure 1.

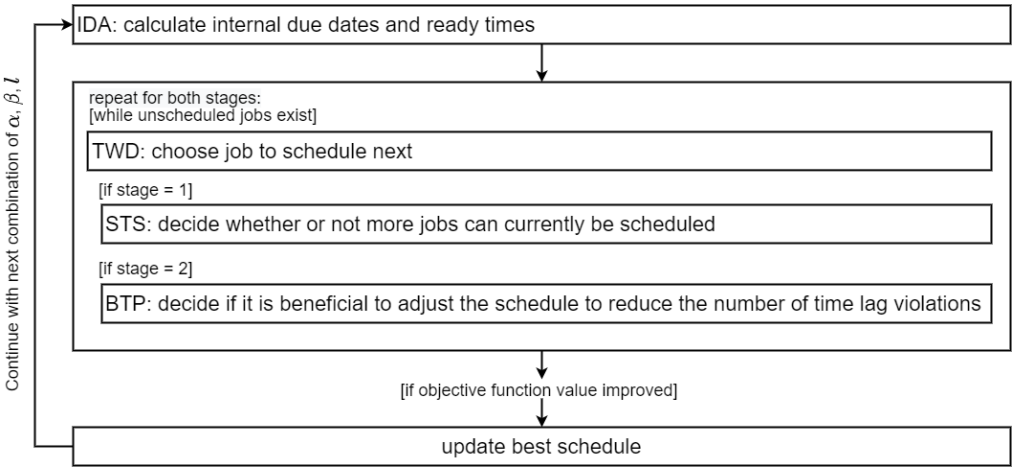

Figure 1: Procedures of the heuristic scheduling framework.

It is crucial to determine suitable priority indices for the operations, since they are used within the TWD procedure. Moreover, the TWD and STS procedures need to be parameterized with respect to the properties

of the jobs. To accomplish this in an automated manner, we use GP with the chromosome design $(I_1, I_2, \Delta_{1,max}, \Delta_{2,max}, \delta)$. Here, I_s are the priority indices discovered by GP, and $\Delta_{s,max}$ is the value to parameterize the TWD procedure for stage s. The δ quantity is used to parameterize the STS. For the GP approach, the priority indices are represented as rooted expression trees that are not limited by a fixed height, allowing the creation of priority indices of variable length (Zhang et al. 2024). Each node in a tree contains either a function or a terminal. By traversing the tree, the corresponding priority index is obtained and a computable expression is created. This expression is then used to calculate the index values of the jobs. The $\Delta_{s,max}$ and δ quantities are represented by an array. The set of terminals used to develop the priority indices consists of both tardiness- and time lag-related terminals and is summarized in Table 1.

Table 1: Terminals used by the GP approach.

Terminal	Description
d	(internal) due date of the current job
r	(internal) ready time of the current job on the current stage
p	processing time of the current job on the current stage
q	stage-dependent slack for the current job to meet the due date
w	weight of the current job
t	current time
pt	average processing time of all jobs on the current stage
$\overline{pt}$	average processing time of the remaining jobs on the current stage
v	length of the time lag of the current job on the current stage or d
g	slack $g := e^{-(t_j - t)^+ / \overline{pt}}$ for the time lag of the current job on the current stage or L
$\bar{g}$	slack $\bar{g} := t - t_j$ for the time lag of the current job j on the current stage or $-\pi$
C	constant value from $\{1, 2, \ldots, 9\}$

The terminal v only takes the size of a time lag if the current job has a time lag on the current stage that can still be fulfilled, otherwise v takes the value of the terminal d. Under the same conditions, the terminals g and $\bar{g}$ evaluate to the time lag-related values, otherwise they take the value L and $-\pi$, respectively, where L is an infinitesimal small positive number, and π is the penalty value when a time lag is violated. Depending on the stage, t_j takes the value of either t_{1j} or t_{12j}.

The set of functions used by GP are addition, subtraction, multiplication, protected division ($a/b = 1$ if $b = 0$), maximum and minimum of two values, powers, positive part a^+, negative value, and exponential function (exp (a)), where a and b are placeholders for the operands. To develop I_s, we use a subtree crossover and a swap tree mutation as genetic operators (Schorn and Mönch 2024). As both indices need to work well together, they are developed in an alternating manner. Starting from the first stage, the stage for which the index is developed changes after each generation. The values of $\Delta_{s,max}$ are developed together with the corresponding indices. A one-point crossover operation (Michalewicz 1996) is applied. When mutation is performed, the value of $\Delta_{s,max}$ is taken from the parent chromosome, randomly increased or decreased by one, and then passed on to the child chromosome. The same operators are applied for δ, but the value of δ is only changed by genetic operators when developing the indices for the first stage.

Developing suitable chromosomes is divided into a training and a validation phase. During the training phase, the chromosomes are trained on a given set of problem instances. By providing the priority indices and the values to parameterize the TWD and STS procedures, the resulting schedules can be used to compute the performance measure values and use them as fitness values of the chromosomes. The genetic operators are applied based on the fitness values, and new chromosomes are created for the next generation. The training phase terminates when either a pre-defined number of generations or a prescribed CT is reached. To validate the performance of the ten best-performing chromosomes, they are applied in the

validation phase on a larger set of problem instances that have the same properties as the set of instances used in the training phase. After the training phase and the validation phases, we obtain the lowest objective function value $OFV := TWT + \pi \cdot TV$ achieved in the validation phase, where TV is the number of time lag violations. We then extract the values from the chromosome that achieved the OFV values and store them, along with the OFV value and the properties of the solved problem instances in a dataset TS. The set is enriched with each training phase and the subsequent validation phases performed. Solving new problem instances based on the created dataset is called the evaluation phase. Here, it is first searched for an entry in TS whose instance properties are close to the properties of the new instance to parameterize the framework with respect to the properties of the new instance.

4 COMPUTATIONAL EXPERIMENTS

4.1 Design of Experiments and Implementation Issues

Randomly generated problem instances with different properties are used to train the GP. We expect that the performance of the different approaches depends on the ready time and due date setting and the presence of time lags. The applied problem instance generation scheme is summarized in Table 2. Here, $U[a,b]$ and $DU[a,b]$ refer to a continuous uniform distribution over $[a,b]$ and to a discrete uniform distribution over the set of integers $\{a,\dots,b\}$, respectively.

Table 2: Problem instance generation scheme to train the GP.

Factor	Level	Count
m_1, m_2	$m_1, m_2 = 4$	1
n	$n = 100$	1
F	$F = 4$	1
p_{1j}, p_{2j}	$p_{1j}, p_{2j} \in \{2,4,10,16,20\}$ with probabilities $\{0.2, 0.2, 0.3, 0.2, 0.1\}$, respectively	1
w_j	$w_j \sim U[0,1]$	1
r_j	$r_j \sim DU\left[0, \left\lfloor \varrho \left(\frac{1}{m_1} \sum_{j=1}^{n} p_{1j} + \frac{1}{m_2} \sum_{j=1}^{n} p_{2j} \right) \right\rfloor\right]$	1
ϱ	$\varrho \in \{0.4, 0.6, 0.8\}$	3
d_j	$d_j := r_j + FF(p_{1j} + p_{2j})$	1
FF	$FF \in \{0.6, 0.8, 1.0, 1.2\}$	4
t_{1j}, tp_1	$t_{1j} - r_j \sim DU[1/n \sum_{j=1}^{n} p_{1j}, 1/m_1 \sum_{j=1}^{n} p_{1j}]$ with probability $tp_1 \in \{0.1, 0.2\}$, ∞ otherwise	2
t_{12j}, tp_{12}	$t_{12j} := 2.5 p_{1j}$ with probability $tp_{12} \in \{0.2, 0.4, 0.6\}$, ∞ otherwise	3

Due to the defined levels, a total of 72 different factor combinations is considered. For each combination, 25 independent problem instances are generated. Five instances per combination are solved in the training phase, while the remaining 20 instances are used for validation (Schorn and Mönch 2024). To appropriately penalize time lag violations, we set $\pi = 200$.

Given the instance generation scheme from Table 2, each entry in TS consists of tuples $(OFV, I_1, I_2, \Delta_{1,max}, \Delta_{2,max}, \delta, \varrho, FF, tp_1, tp_{12})$, where ϱ is a distribution parameter for the ready times, FF denotes the flow factor, and tp_1 and tp_{12} are the probabilities of initial and regular time lags, respectively. When a new instance is solved during the evaluation phase, we estimate $\hat{\varrho} :=$

$\frac{2}{n}\sum_{j=1}^{n} r_j / \left(\frac{1}{m_1}\sum_{j=1}^{n} p_{1j} + \frac{1}{m_2}\sum_{j=1}^{n} p_{2j}\right)$, $\widehat{FF} \coloneqq \frac{1}{n}\sum_{j=1}^{n} \frac{d_j - r_j}{p_{1j}+p_{2j}}$, $\widehat{tp}_1 \coloneqq \frac{1}{n}\left|\{j \mid 1 \leq j \leq n, t_{1j} \neq \infty\}\right|$, and $\widehat{tp}_{12} \coloneqq \frac{1}{n}\left|\{j \mid 1 \leq j \leq n, t_{12j} \neq \infty\}\right|$. Since only a short amount of CT is needed to solve a new instance during the evaluation phase, we select the five best matching entries from TS with $\tau \coloneqq \underset{i \in TS}{\operatorname{argmin}}\left(\max\left\{|\hat{\varrho} - \varrho_i|, \left|\widehat{FF} - FF_i\right|, |\widehat{tp}_1 - tp_{1i}|, |\widehat{tp}_{12} - tp_{12i}|\right\}\right)$. The parameter values $\left(I_1, I_2, \Delta_{1,max}, \Delta_{2,\max}, \delta\right)$ of the entries labeled by τ are then used to parameterize the scheduling framework to solve the new instance.

The GP operates on a population with 400 chromosomes and with a crossover and mutation probability of 80% and 20%, respectively. The replacement rate is 50%. The tree structures can have a maximum height of four when created and can grow to a maximum height of eight during the process. Each node can have a maximum of three child nodes. The GP is trained for 100 generations or 7200s of CT, whichever is reached first. To reduce the computational effort required to train the GP, some procedures are modified for the training phase based on insights from preliminary experiments. First, we refrain from using the IDA procedure, since it is sufficient to apply the IDA procedure during validation. To obtain internal due dates for the training phase, the slack of each job is evenly distributed over both stages. Moreover, the search space of the GP is restricted by defining lower and upper bounds for $\Delta_{s,max}$ and δ, ensuring that genetic operators are only applied if they do not result in a violation of $2 \leq \Delta_{s,max} \leq 12$ and $4 \leq \delta \leq 16$. The chromosomes are initialized using $\Delta_{s,max} \sim DU[2,12]$ and $\delta \sim DU[8,12]$. Finally, only one value $\Delta \sim DU[0, \Delta_{s,max}]$ is considered during the TWD procedure.

To apply the IDA procedure during the validation and evaluation phases, the values for α and β are taken independently from the grid $\alpha, \beta \in \{0.0, 0.25, \ldots, 1\}$. Three iterations are performed, i.e. $l_{max} \coloneqq 3$. After conducting preliminary experiments, the ATC dispatching rule used in the reference heuristic is parameterized with $\gamma \coloneqq 0.5$. Values for κ are taken from the grid $\kappa = 0.5k$ for $k = 1, \ldots, 10$. For each value of κ, three iterations are performed. The BRKGA is performed for a CT of 600s. The maximum CT per MILP instance is set to 30s, and each MILP instance takes a maximum of $\tilde{n} \coloneqq 10$ jobs into account.

We conduct a series of experiments to assess the performance of the heuristic scheduling framework during the evaluation phase. We start by evaluating the performance under conditions optimal for GP by applying the framework to 72 new problem instances with the same properties as the instances used to train the GP. To evaluate the robustness, several experiments with 100 problem instances each are performed. First, we create instances with a scheme based on distributions instead of fixed values by using $n = x \cdot F, x \sim DU[15, 35], \varrho \sim U[0.4, 0.8], FF \sim U[0.6, 1.2], tp_1 \sim U[0.1, 0.2]$, and $tp_{12} \sim U[0.2, 0.6]$. All other values remain unchanged. In addition, we use this scheme to generate instances with much tighter ready times than the instances used for training by setting $\varrho \sim U[0.1, 0.2]$. Furthermore, we perform experiments with a different number of families and widely varying but fixed processing times per family by adjusting the generation scheme using $F \sim DU[2,8]$ and $x \sim DU[\lceil 60/F \rceil, \lceil 140/F \rceil]$. The processing times for the operations executed on the first and second stage are set as $p_{1f} \coloneqq (2, 20, 4, 16, 2, 20, 4, 16)$ and $p_{2f} \coloneqq (4, 16, 2, 20, 4, 16, 2, 20)$ for $f = 1, \ldots, 8$, respectively.

All algorithms are coded in the C++ programming language. The GP is implemented using the GaLib framework (Wall 2025). The BRKGA is coded using the brkgaAPI framework (Tosso and Resende 2025). IBM ILOG CPLEX Optimization Studio 12.7 is used for solving the MILPs. The computational experiments are performed on a PC with an Intel Core i7-10700 with 2.90 GHz and 32 GB of RAM.

4.2 Results

To assess the performance, we report the average TWT value per solved instance obtained with the heuristic scheduling framework and the reference heuristic, which will be denoted as HSF and BRKGA+MILP in the following tables, respectively. In addition, we also report the percentage improvement achieved by the framework over the reference heuristic. The results for the 72 problem instances with the same properties as the instances used to train the GP are shown in an aggregated form in Table 3.

The results shown in Table 3 indicate that the tightness of the ready times, i.e. the value of ϱ, has the highest impact on the results. The HSF achieves the highest improvements over the reference heuristic for instances with high ϱ values, suggesting that more loose ready times are beneficial for the framework.

Table 3: Results for problem instances with the same properties as the instances used for training.

Factor/Level	HSF	BRKGA+MILP	Improvement
$\varrho = 0.4$	422.18	402.47	-4.90%
$\varrho = 0.6$	129.57	186.62	30.57%
$\varrho = 0.8$	49.69	79.50	37.50%
$FF = 0.6$	286.43	307.60	6.88%
$FF = 0.8$	209.64	247.13	15.17%
$FF = 1.0$	168.01	198.57	15.39%
$FF = 1.2$	137.84	138.16	0.23%
$tp_1 = 0.1$	200.37	221.15	9.40%
$tp_2 = 0.2$	200.59	224.58	10.68%
$tp_{12} = 0.2$	195.98	220.09	10.95%
$tp_{12} = 0.4$	202.80	227.88	11.00%
$tp_{12} = 0.6$	202.65	220.63	8.15%
Overall	200.47	222.86	10.04%

Moreover, we can observe that a moderate value for the flow factor, i.e. $FF \in \{0.8, 1.0\}$, yields notably higher improvements compared to both lower $FF = 0.6$ and higher $FF = 1.2$ values. The probabilities of initial tp_1 and regular time lags tp_{12} have only minor effects on the results. Under consideration of all solved instances, an average improvement of 10.04% per instance is achieved. The results for the problem instances generated using perturbed distributions are shown in Table 4.

Table 4: Results for problem instances generated with distributions.

Factor/Level	HSF	BRKGA+MILP	Improvement
$n \leq 80$	278.91	298.44	6.54%
$80 < n \leq 100$	373.19	395.82	5.72%
$100 < n \leq 120$	456.26	495.64	7.95%
$n > 120$	513.84	545.03	5.72%
$\varrho \leq 0.5$	751.60	776.57	3.22%
$0.5 < \varrho \leq 0.6$	388.98	425.36	8.55%
$0.6 \leq \varrho \leq 0.7$	244.13	267.54	8.75%
$\varrho > 0.7$	197.57	231.13	14.52%
$FF \leq 0.8$	524.21	543.12	3.48%
$0.8 < FF \leq 1.0$	428.55	471.65	9.14%
$FF > 1.0$	263.04	288.32	8.77%
$tp_1 \leq 0.15$	468.69	499.55	6.18%
$tp_1 > 0.15$	343.87	370.11	7.09%
$tp_{12} \leq 0.4$	426.06	453.17	5.98%
$tp_{12} > 0.4$	395.22	425.37	7.09%
Overall	431.60	461.80	6.54%

From Table 4 we observe that the number of jobs n has no significant impact on the results, i.e., the framework is robust with respect to problem instance sizes. Regarding the tightness of the ready times, we can see in accordance with the results presented in Table 3 that a large value of ϱ is beneficial for the framework. The same can be observed for the flow factor FF, as larger improvements are achieved for instances with $FF > 0.8$. Again, no major impact on the results depending on the probabilities for initial tp_1 and regular time lags tp_{12} can be observed. However, the framework achieves slightly higher improvements over the reference heuristic for instances including more jobs with time lags. In contrast to solving the instances with the same properties as the instances used for training, selecting the five best matching entries leads to a notable improvement in solution quality for the instances created with distributions, as no exact matching entries exist in the dataset TS. Taking all solved instances into account, an average improvement of 6.54% over the reference heuristic is achieved by the framework. To visualize the properties of the solved problem instances, we show the values of n, ϱ, FF, tp_1 and tp_{12} for each of the solved instances in Figure 2.

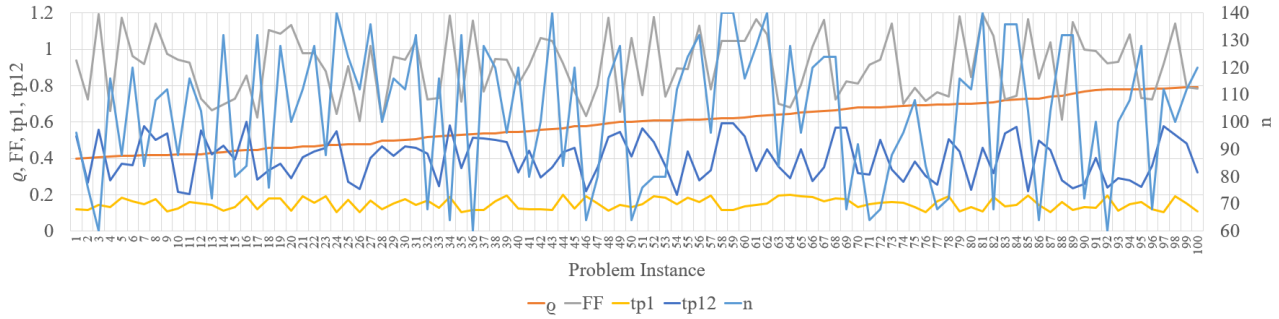

Figure 2: Properties of the problem instances created with distributions.

The results presented in Table 4 and the properties illustrated in Figure 2 show that the framework is robust and therefore able to create high-quality schedules for a stream of incoming problem instances with different properties. For the special case with tighter ready times, the framework achieves an average TWT value of 2165.08 per instance. In comparison, the reference heuristic achieves a lower average TWT value of 2092.03, which represents a negative improvement of -3.49% for the framework relative to reference heuristic. The performance of the framework is only slightly worse despite the fact that the GP is not trained on instances with such tight ready times. However, this highlights the importance of creating a sufficiently diverse dataset to make the framework adaptable and successfully applicable to problem instances with different properties. The results for the experiments with different numbers of families and widely varying but fixed processing times per family are shown in Table 5.

Table 5: Results for problem instances with different numbers of families.

Factor/Level	HSF	BRKGA+MILP	Improvement
$F = 2$	358.00	399.59	10.41%
$F = 3$	190.14	211.41	10.06%
$F = 4$	329.80	341.70	3.48%
$F = 5$	180.79	193.55	6.59%
$F = 6$	364.03	370.14	1.65%
$F = 7$	378.50	386.59	2.09%
$F = 8$	379.29	385.73	1.67%
Overall	320.26	335.29	4.48%

From Table 5 we can observe that a small number of different families is beneficial for the framework. Interestingly, the highest improvements per instance compared to the reference are achieved for $F \in \{2, 3\}$, even though the GP is trained on instances characterized by $F = 4$. With a larger number of families, i.e. with $F \in \{6, 7, 8\}$, the improvement compared to the reference heuristic decreases.

Considering all problem instances solved during the experiments, neither the application of the framework nor the reference heuristic result in any time lag violations. Due to the applied error function, the parameters $\varrho, FF, tp_1, tp_{12}$ are the decisive selection criteria in 29%, 33%, 14% and 24% of the cases, respectively. Training the GP twice on the same factor combinations and storing the best chromosomes for each factor combination in the dataset shows only small improvements in solution quality. Training the GP with problem instances of different sizes $n \in \{80, 100, 120\}$ also causes only a small effect. Because the GP is trained on multiple factor combinations in parallel, the training takes around 8.4 hours or 30240 seconds. The average CT required by the framework to solve a new instance during the evaluation phase is 6.11 seconds. Solving a new instance with the reference heuristic takes around 900 seconds on average, depending on the number of jobs.

5 CONCLUSIONS AND FUTURE WORK

A two-stage flexible flow-shop scheduling problem with maximal time lags was studied. We compared the performance of a heuristic scheduling framework based on GP with a job-based decomposition approach based on a BRKGA and a MILP-based improvement phase. We observed by designed computational experiments that the GP approach is able to solve problem instances much faster than the decomposition approach after enough time is spent for training the GP and thus creating a sufficiently diverse dataset. The GP-based approach leads to high-quality schedules and is robust with respect to changes in the properties of the instances used for training, validation, and evaluation.

There are several directions for future research. First of all, we are interested in extending the two approaches from flow shops to job shops. This requires that discrete-event simulation must be used to assess the schedules under process uncertainty. It is also interesting to design algorithms that take into account overlapping maximal time lags.

ACKNOWLEDGMENTS

This research was supported by the European Commission and the German Authorities through the AIMS5.0 project. The project AIMS5.0 is supported by the Chips Joint Undertaking and its members, including the top-up funding by National Funding Authorities from involved countries under grant agreement no. 101112089. The authors gratefully acknowledge this financial support. The work of the second author was also partially supported by the DFG, project grant MO 1021/9-1.

REFERENCES

Branke, J., S. Nguyen, C. W. Pickardt, and M. Zhang. 2016. “Automated Design of Production Scheduling Heuristics: A Review”. *IEEE Transactions on Evolutionary Computation* 20(1):110–124.

Braune, R., F. Benda, K. F. Doerner, and R. F. Hartl. 2022. “A Genetic Programming Learning Approach to Generate Dispatching Rules for Flexible Shop Scheduling Problems”. *International Journal of Production Economics* 243:108342.

Cailloux, J., and L. Mönch. 2019. “Scheduling Jobs in Flexible Flow Shops with Batching and Time Constraints”. In *Proceedings MISTA*, December 12th-15th, Ningbo, China, 603–607.

Ferreira, C., G. Figueira, and P. Amorim, P. 2022. “Effective and Interpretable Dispatching Rules for Dynamic Job Shops via Guided Empirical Learning”. *Omega* 111:102643.

Gonçalves, J. F., and M. G. C. Resende. 2011. “Biased Random-key Genetic Algorithms for Combinatorial Optimization”. *Journal of Heuristics* 17(5):487–525.

Han, J.-H., and J.-Y. Lee. 2023. “Scheduling for a Flow Shop with Waiting Time Constraints and Missing Operations in Semiconductor Manufacturing”. *Engineering Optimization* 55(10):1742–1759.

Hildebrandt, T., D. Goswami, and M. Freitag, M. 2014. “Large-scale Simulation-based Optimization of Semiconductor Dispatching Rules”. In *2014 Winter Simulation Conference (WSC)*, 2580–2590 https://doi.org/10.1109/WSC.2014.7020102.

Jung, C., D. Pabst, M. Ham, M. Stehli, and M. Rothe. 2014. "An Effective Problem Decomposition Method for Scheduling of Diffusion Processes Based on Mixed Integer Linear Programming". *IEEE Transactions on Semiconductor Manufacturing* 27(3):357–363.

Klemmt, A., and L. Mönch. 2012. "Scheduling Jobs with Time Constraints between Consecutive Process Steps in Semiconductor Manufacturing". In *2012 Winter Simulation Conference (WSC)*, 2173-2182 https://doi.org/10.1109/WSC.2012.6465235.

Knopp, S. 2016. *Complex Job-Shop Scheduling with Batching in Semiconductor Manufacturing*. PhD thesis, l'École des Mine de Saint-Étienne, Center of Microelectronics in Provence.

Kopp, D., M. Hassoun, A. Kalir, and L. Mönch. 2020. "Integrating Critical Queue Time Constraints Into SMT2020 Simulation Models". In *2020 Winter Simulation Conference (WSC)*, 1813-1824 https://doi.org/10.1109/WSC48552.2020.9383889.

Kück, M., E. Broda, M. Freitag, T. Hildebrandt, and E. M. Frazzon,. 2017. "Towards Adaptive Simulation-based Optimization to Select Individual Dispatching Rules for Production Control". In *2017 Winter Simulation Conference (WSC)*, 3852–3863 https://doi.org/10.1109/WSC.2017.8248096.

May, M. C., J. Oberst, and G. Lanza. 2024. "Managing Product-inherent Constraints with Artificial Intelligence: Production Control for Time Constraints in Semiconductor Manufacturing". *Journal of Intelligent Manufacturing* 35:4259–4276.

Michalewicz, Z. 1996. *Genetic Algorithms + Data Structures = Evolution Programs*. 3rd ed., Berlin: Springer.

Mönch, L., H. Balasubramanian, J. W. Fowler, and M. E. Pfund. 2005. "Heuristic Scheduling of Jobs on Parallel Batch Machines with Incompatible Job Families and Unequal Ready Times". *Computers & Operations Research* 32(11):2731–2750.

Mönch, L., J. W. Fowler, S. Dauzère-Pérès, S. J. Mason, and O. Rose. 2011. "A Survey of Problems, Solution Techniques, and Future Challenges in Scheduling Semiconductor Manufacturing Operations". *Journal of Scheduling* 14(6):583–599.

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

Mönch, L., and J. Zimmermann. 2004. "Improving the Performance of Dispatching Rules in Semiconductor Manufacturing by Iterative Simulation". In *2004 Winter Simulation Conference (WSC)*, 1881–1887 https://doi.org/10.1109/WSC.2004.1371544.

Qin, M., R. Wang., Z. Shi, L Liu, and L. Shi. 2021. "A Genetic Programming-based Scheduling Approach for Hybrid Flow Shop with a Batch Processor and Waiting Time Constraint". *IEEE Transactions on Automation Science and Engineering* 18(1):94–105.

Scholl, W., and J. Domaschke. 2000. "Implementation of Modeling and Simulation in Semiconductor Wafer Fabrication with Time Constraints Between Wet Etch and Furnace Operations". *IEEE Transactions on Semiconductor Manufacturing* 13(3):273–277.

Schorn, D., and L. Mönch. 2024. "Learning Priority Indices for Energy-aware Scheduling of Jobs on Batch Processing Machines". *IEEE Transactions on Semiconductor Manufacturing* 37(1):3–15.

Schorn, D., and L. Mönch. 2025. "A Genetic Programming Approach to Solve Flexible Flow-shop Scheduling Problems with Batching and Time Constraints". Submitted for publication.

Tan, Y., L. Mönch, and J. W. Fowler 2018. "A Hybrid Scheduling Approach for a Two-stage Flexible Flow shop with Batch Processing Machines *Journal of Scheduling* 21(2):209–226.

Toso, R. F., and M. G. C. Resende. 2025. A C++ Application Programming Interface for Biased Random-key Genetic Algorithms. http://mauricio.resende.info/doc/brkgaAPI.pdf. Last accessed 2025, April 28th.

Wall, M. 2025. GaLib: A C++ Library for Genetic Algorithm Components. https://lancet.mit.edu/ga/. Last accessed 2025, April 28th.

Yugma, C., S. Dauzère-Pérès, S., C. Artigues, A., Derreumaux, and O. Sibille. 2012. "A Batching and Scheduling Algorithm for the Diffusion Area in Semiconductor Manufacturing"**.** *International Journal of Production Research* 50(8):2118–2132.

Zhang, T., F. Pappert, and O. Rose. 2016. "Time Bound Control in a Stochastic Dynamic Wafer Fab". In *2016 Winter Simulation Conference (WSC)*, 2903-2911 https://doi.org/10.1109/WSC.2016.7822325.

Zhang, F., Y. Mei, S. Ngyuen, and M. Zhang. 2024. "Survey on Genetic Programming and Machine Learning Techniques for Heuristic Design in Job Shop Scheduling". *IEEE Transactions on Evolutionary Computing* 8(1):147–167.

AUTHOR BIOGRAPHIES

DANIEL SASCHA SCHORN is currently a PhD student in information systems at the University of Hagen. He received bachelor and master degrees in Information Systems from the University of Hagen. His research interests are scheduling for semiconductor manufacturing and discrete-event simulation. His email address is daniel-sascha.schorn@fernuni-hagen.de.

LARS MÖNCH is full professor of Computer Science at the Department of Mathematics and Computer Science, University of Hagen where he heads the Chair of Enterprise-wide Software Systems. He holds M.S. and Ph.D. degrees in Mathematics from the University of Göttingen, Germany. After his Ph.D., he obtained a habilitation degree in Information Systems from Technical University of Ilmenau, Germany. His research and teaching interests are in information systems for production and logistics, simulation, scheduling, and production planning. His email address is Lars.Moench@fernuni-hagen.de. His website is https://www.fernuni-hagen.de/ess/team/lars.moench.shtml.