RAIRO-Oper. Res. 52 (2018) 1329–1350
https://doi.org/10.1051/ro/2018029

RAIRO Operations Research
www.rairo-ro.org

HYBRID ALGORITHMS FOR THE EARLINESS–TARDINESS SINGLE-MACHINE MULTIPLE ORDERS PER JOB SCHEDULING PROBLEM WITH A COMMON DUE DATE

Jens Rocholl and Lars Mönch*

Abstract. In this paper, we study an earliness–tardiness scheduling problem for a single machine that is motivated by process conditions found in semiconductor wafer fabrication facilities (wafer fabs). In modern 300-mm wafer fabs, front opening unified pods (FOUPs) transfer wafers. The number of FOUPs is limited to avoid a congestion of the Automated Material Handling System. Several orders can be grouped in one FOUP. A nonrestrictive common due date for all the orders is assumed. Only orders that belong to the same family can be processed together in a single FOUP at the same time. We present a Mixed Integer Linear Programming (MILP) formulation for this problem. Moreover, we show that this scheduling problem is NP-hard. We propose several simple heuristics based on dispatching rules and assignment strategies from bin packing. Moreover, genetic algorithms are designed that assign the orders to the set of early and tardy orders, respectively. In addition, a random key genetic algorithm (RKGA) is described that proposes order sequences. The different algorithms are hybridized with job formation and sequencing heuristics. A more specialized algorithm that is based on the generalized assignment problem is presented for the special case of a single order family. Results of computational experiments based on randomly generated problem instances are presented. They demonstrate that the genetic algorithms perform well with respect to solution quality and computing time under a broad range of experimental conditions.

Mathematics Subject Classification. 90B20, 90B35, 68R01.

Received May 31, 2017. Accepted April 27, 2018.

1. Introduction

Semiconductor manufacturing deals with manufacturing integrated circuits (ICs). The manufacturing process starts from raw wafers, thin discs made of silicon or gallium arsenide. The diameter of such a disc is 200- or 300 mm in today's semiconductor wafer fabrication facilities (wafer fabs). Up to several thousands of ICs can be placed onto a single wafer. The ICs are built up layer by layer onto the wafer surface by means of complicated and often highly automated manufacturing processes. The wafer fabrication part of semiconductor manufacturing is carried out in wafer fabs. It is characterized by a couple of specific process restrictions such as reentrant process flows that consist of several hundreds of process steps, sequence-dependent setup times that are often much longer than the processing times, a mix of single wafer, lot, and batch processing, *i.e.*, wafers of several jobs

Keywords and phrases: Scheduling, multiple orders per job, common due date, semiconductor manufacturing.

Department of Mathematics and Computer Science, University of Hagen, 58097 Hagen, Germany.

* Corresponding author: Lars.Moench@fernuni-hagen.de

can be processed at the same time on the same machine, and expensive machines that are organized in machine groups (*cf.* [27]). Production control activities are mainly supported by often sophisticated dispatching rules. Until recently, scheduling methods for wafer fabs seemed to be too costly compared to dispatching approaches. However, with the recent dramatic increase in computer efficiency, scheduling methods have become more competitive (*cf.* [27]).

Due to the increasing automation pressure, automated material handling systems (AMHS) are very common in modern wafer fabs (*cf.* [1, 3, 29]). A FOUP is a standard unit of AMHS-based job transfer between machines where a FOUP is a container that is able to hold up to 25-wafer lots of 300-mm wafers in an inert, nitrogen atmosphere (*cf.* [6]). FOUPs are the moving entities in a wafer fab that might visit the machines of the different machine groups several times. The number of FOUPs is a restriction in wafer fabs since FOUPs are expensive and a large number of them might lead to a congested AMHS. Because of the combination of decreased line width and increased area per wafer in 300-mm wafer fabs, the number of wafers that is needed to fill the ICs of a certain customer is smaller compared to the situation found in mature 200-mm wafer fabs. Therefore, there is quite often a need to group orders of different customers into one FOUP. The resulting entity is called a job to go conform to the scheduling literature. The jobs have to be scheduled on the machines of a wafer fab after the grouping decision is made. We refer to the resulting problems as multiple orders per job (MOJ) scheduling problems (*cf.* [25]).

In this paper, we consider a single-machine scheduling problem. A lot processing environment is assumed, *i.e.*, all orders that are assigned to a job have the same processing time. This time is the processing time of the job. All orders have a common due date that is "unrestrictively late" (or nonrestrictive). The common due date assumption is motivated by the situation found in memory business where it is likely that a large number of chips have the same external (loose or tight) due date and hence the same internal due date for the corresponding wafer fab operations. The occurrence of base wafers in wafer fabs provides a second motivating example. The notion of base wafers refers to preprocessed wafer, *i.e.*, all layers up to the metal layer are already processed [28]. Base wafers are held on stock in a wafer bank for fulfilling specific customer requests based on contractual obligations. In this situation, the due dates of almost all final chips are the same and often loose since the company has some flexibility to fulfill the contracts, therefore the same holds for the related internal due dates in the wafer fab.

We are interested in minimizing the sum of the absolute deviations of the completion times of all orders from the common due date (earliness–tardiness). The earliness is related to inventory holding whereas tardiness refers to customer dissatisfaction. The earliness–tardiness (E/T) measure is important in just-in-time environments. To the best of our knowledge, a MOJ scheduling problem with the E/T measure and a common due date of the orders is not considered so far in the literature.

The paper is organized as follows. The MOJ scheduling problem is described in Section 2. A MILP formulation is presented. Moreover, related work is discussed in this section. Structural properties of optimal solutions are established in Section 3. In addition, the NP-hardness of the scheduling problem is shown. Different heuristic and exact approaches are designed in Section 4. The results of computational experiments are presented in Section 5. Finally, conclusions and future research directions are discussed in Section 6.

2. Problem formulation and discussion of related work

We start by describing the problem setting in Section 2.1. A MILP formulation for the MOJ scheduling problem is presented in Section 2.2. We discuss related work in Section 2.3.

2.1. Problem setting

A single machine is considered. There are N orders that belong to L incompatible families. We assume that $N_l, l = 1, \ldots, L$ orders are in family l. We have $\sum_{l=1}^{L} N_l = N$. We denote the set of all orders by O and the set of orders that belong to family l by O_l. Each order o has a size s_o that is given as the number of wafers. Only orders belonging to the same family can be grouped into one job, *i.e.* a FOUP. A split of orders into suborders is not allowed. We have F FOUPs. Once the machine is started to process a job it cannot be interrupted. Since

FOUPs are standardized, it is reasonable to assume that all FOUPs have the same capacity of K wafers, *i.e.* items. We denote by $f(o)$ the family of order o. All orders are available at time $t = 0$. The orders have a common due date d that is unrestrictively late (or nonrestrictive), *i.e.*, we assume

$$\sum_{o=1}^{N} p_o \leq d, \tag{2.1}$$

where p_o is the processing time of order o. All orders o that belong to the same family $f(o) = l$ have the same processing time P_l. We consider a lot processing environment where all wafers of a job will be processed together. Therefore, the processing time of a job is determined by the family of the orders that form the job, *i.e.*, it is equal to P_l if l is the family of these orders. Let C_o be the completion time of order o. Due to the lot processing environment, the completion time of all orders that form a job is identical. We are interested in minimizing the E/T performance measure of the N orders. The earliness of order o is given by $E_o := (d - C_o)^+$ and the tardiness by $T_o := (C_o - d)^+$. Here, we use $x^+ := \max(x, 0)$ for abbreviation. The E/T performance measure is given by

$$E/T := \sum_{o=1}^{N} (E_o + T_o) = \sum_{o=1}^{N} |C_o - d|. \tag{2.2}$$

Using the three-field notation from scheduling theory [11], the scheduling problem can be represented as

$$1|moj\,(lot)\,, incompatible, d_o \equiv d|E/T, \tag{2.3}$$

where $moj\,(lot)$ refers to a MOJ setting with lot processing, and *incompatible* indicates the different order families. Finally, the nonrestrictive common due date d is represented by $d_o \equiv d$.

The following two decisions have to be made when an instance of problem (2.3) is solved:

1. Job formation: It has to be decided which orders have to be grouped together into a single job.
2. Job sequencing: A sequence of the already formed jobs has to be determined.

We will see that the first decision makes the scheduling harder to solve than its conventional counterpart $1|d_o \equiv d|E/T$ which can be efficiently solved due to Kanet [16].

An example instance with $L = 3$ incompatible families is shown in Figure 1. Here, we assume that the common due date d is nonrestrictive, while the processing time of the orders of the first, second, and third family is 1, 2, and 4, respectively. The capacity of a job is six items. The different order families are indicated by different colors and shadings.

We see that five jobs are formed and sequenced around the common due date. Note that a concrete value of the common due date d is not important as far it is large enough since there is one job that is completed at d. All the completion times of orders in the early and tardy order set are relative to d in this situation. The feasible schedule shown in Figure 1 has an objective function value of $E/T = 1 \cdot 1 + 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 7 = 30$.

2.2. MILP formulation

Next, we present the MILP formulation for problem (2.3). Without loss of generality, we assume that we have a fixed sequence of the jobs. We start by presenting indices and sets:

$1 \leq j, k \leq F$: jobs
$o = 1, \ldots, N$: orders
$l = 1, \ldots, L$: order families.

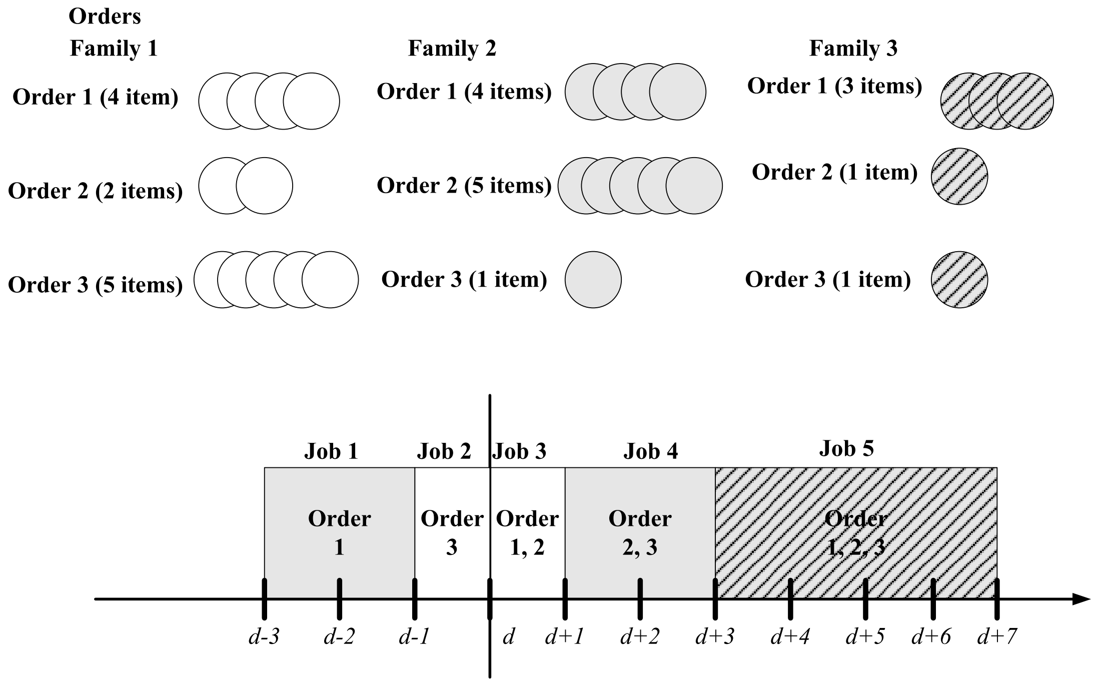

FIGURE 1. Example of a feasible schedule.

The following decision variables are used in the model:

$$x_{oj} := \begin{cases} 1, & \text{if order } o \text{ belongs to job } j \\ 0, & \text{otherwise} \end{cases}$$

$$y_{jl} := \begin{cases} 1, & \text{if order family } l \text{ is assigned to job } j \\ 0, & \text{otherwise} \end{cases}$$

e_j : completion time of job j
C_o: completion time of order o
E_o: earliness of order o
T_o: tardiness of order o.

The model is based on the following parameters:

s_o: size of order o (in wafers)
d: common due date
K: capacity of a FOUP (in wafers)
p_l: processing time of each order of order family l
G: very large number.

The model can be described as follows:

$$\min \sum_{o=1}^{N} (E_o + T_o) \tag{2.4}$$

subject to

$$\sum_{j=1}^{F} x_{oj} = 1,\ o = 1, \ldots, N, \tag{2.5}$$

$$\sum_{o=1}^{N} s_o x_{oj} \leq K,\ j=1,\ldots,F, \tag{2.6}$$

$$\sum_{l=1}^{L} y_{jl} = 1,\ j=1,\ldots,F, \tag{2.7}$$

$$x_{oj} - y_{jf(o)} \leq 0,\ o=1,\ldots,N, j=1,\ldots,F, \tag{2.8}$$

$$e_j + p_l y_{kl} \leq e_k,\ o=1,\ldots,N, j=1,\ldots,F, k=1,\ldots,F, j<k, l=1,\ldots,L, \tag{2.9}$$

$$e_j - G\left(1-x_{oj}\right) \leq C_o,\ o=1,\ldots,N, j=1,\ldots,F, \tag{2.10}$$

$$C_o \leq e_j + G\left(1-x_{oj}\right),\ o=1,\ldots,N, j=1,\ldots,F, \tag{2.11}$$

$$C_o - d \leq T_o,\ o=1,\ldots,N, \tag{2.12}$$

$$d - C_o \leq E_o,\ o=1,\ldots,N, \tag{2.13}$$

$$x_{oj}, y_{jl} \in \{0,1\},\ 0 \leq C_o, e_j, E_o, T_o, o=1,\ldots,N, j=1,\ldots,F, l=1,\ldots,L. \tag{2.14}$$

The objective function (2.4) is the sum of the earliness and tardiness values of the different orders. Constraints (2.5) make sure that each order is assigned to exactly one job. The capacity of the FOUPs is ensured by constraints (2.6). Constraints (2.7) model that exactly one family is assigned to each job. It is expressed by constraints (2.8) that all orders of a certain job belong to the same family. Constraints (2.9) relate the completion time of jobs to the sequence of the jobs and the corresponding processing times. The completion time of the individual orders of a job is derived based on constraints (2.10) and (2.11). Constraints (2.12) and (2.13) provide a linearization of the tardiness and earliness values of the individual orders. The domains of the decision variables are ensured by the constraints (2.14).

2.3. Related work

We discuss related work with respect to MOJ scheduling and to single and parallel machine scheduling problems with a common due date and the E/T measure. We start by MOJ scheduling problems. The MOJ scheduling problems $1|moj\,(lot)\,|TWC$ and $1|moj\,(item)\,|TWC$ are analyzed in Mason *et al.* [21] for the first time. Here, we denote the item processing case by $moj\,(item)$, *i.e.*, the processing time of a job depends on the number of wafers, *i.e.* items, in the job. Moreover, we abbreviate the total weighted completion time performance measure by TWC. Fairly simple heuristics motivated by bin packing problems are formulated. A two-phase MIP-based solution technique for $1|moj\,(lot)\,|TC$ and $1|moj\,(item)\,|TC$, respectively is proposed by Mason and Chen [20] where TC refers to the total completion time of the orders to be scheduled. Two genetic algorithms (GAs) are proposed for the MOJ single-machine scheduling problems $1|r_o, moj\,(lot)\,|TWC$, $1|r_o, moj\,(item)\,|TWC$, $1|r_o, moj\,(lot)\,|TWT$, and $1|r_o, moj\,(item)\,|TWT$ by Qu and Mason [32] where unequal release dates of the orders are indicated by the notation r_o. Here, the abbreviation TWT is used for the total weighted tardiness performance measure. Erramilli and Mason [4] use simulated annealing to solve the problem $1|moj\,(lot)\,, batch|TWT$. Here, the notation *batch* indicates a situation where several jobs can be processed together, *i.e.*, they form a batch. A MOJ single-machine batch problem with incompatible job families is addressed by Erramilli and Mason [5]. Only jobs that belong to the same family can be used to form a batch. Iterative schemes for the problem $1|moj\,(item)\,|TC$ are designed by Tanrisever and Kutanoglu [40]. A branch-and-bound algorithm for the problem $1|moj\,(item)\,|TWC$ is proposed by Sarin *et al.* [34].

Column generation is applied by Jampani and Mason [13] to the problem $Pm|r_o, moj\,(lot)\,|TWC$, where Pm is used for identical parallel machines. Jia and Mason [15] propose several heuristics for the MOJ problem $Pm|r_o, s_{kl}, moj\,(lot)\,|TWC$. Here, s_{kl} refers to sequence-dependent setup times.

The scheduling problems $F2|moj\,(lot)\,|C_{\max}$ and $F2|moj\,(item)\,|C_{\max}$ are discussed by Laub *et al.* [17], where $F2$ is a flow shop with two machines. The makespan objective $C_{\max}$is assumed. Optimal approaches and a heuristic for $F2|moj\,(item)\,|C_{\max}$ are proposed. The problem studied by Laub *et al.* [17] is generalized by Sarin *et al.* [35]. New structural properties are proven. Moreover, tight upper bounds are computed. Jampani *et al.* [14] hybridize ant colony optimization and constraint programming to solve the MOJ scheduling problem $FJm\,|moj\,(lot)\,, r_o, recrc|TWC$, where FJm and $recrc$ represent a complex job shop and reentrant flows, respectively. Mönch *et al.* [26] design and assess several job formation and release strategies for large-scale wafer fabrication facilities using discrete-event simulation.

The single-machine scheduling problems $1|r_o, moj\,(lot)\,|TWC$ and $1|r_o, moj\,(item)\,|TWC$ are studied by Sobeyko and Mönch [36]. A RKGA and a grouping GA (GGA) are designed for these problems. Moreover, Sobeyko and Mönch [37] propose GGAs to solve the four single-machine problems $1|r_o, moj\,(lot)\,|TWT$, $1|r_o, moj\,(item)\,|TWT$, $1|r_o, moj\,(lot)\,|WNTO$, $1|r_o, moj\,(item)\,|WNTO$ where we denote the weighted number of tardy orders performance measure by WNTO.

Next, we discuss related work with respect to common due date scheduling. It is shown by Kanet [16] that the scheduling problem $1|d_j \equiv d|E/T$ can be solved by an efficient exact algorithm. Hall and Posner [12] generalize the setting from Kanet [16] by allowing weights for the jobs. The resulting scheduling problem with the weighted earliness and tardiness measure is NP-hard. Structural results are derived, and several constructive heuristics are proposed. For a survey of common due date-based scheduling research we refer to Gordon *et al.* [10].

Next, we discuss scheduling research with a common due date where a batch processing machine, *i.e.* a p-batching environment, is assumed. The processing time of a batch in a p-batching environment is determined as the longest processing of the jobs that form the batch. Since the objective function is job-based, the jobs in a batch lead to problems that are similar to the scheduling problem discussed by Hall and Posner [12], *i.e.*, the number of jobs in a batch can be interpreted as a weight for the batch. Common due date scheduling problems for batch processing machines are similar to problem (2.3) since grouping decisions have to be made too. Mönch *et al.* [24] discuss the problem $1|p - batch, d_j \equiv d, \Delta|E/T$. A maximum allowable tardiness constraint is considered. Here, Δ denotes the amount of maximum allowable tardiness. They propose a GA that is based on structural properties of optimal solutions. The problem $Pm|p - batch, d_j \equiv d|E/T$ is discussed by Mönch and Unbehaun [23]. Genetic algorithms are proposed. Li *et al.* [18] extend the heuristics from Mönch and Unbehaun [23] to a single-machine situation where unequal job sizes are possible. The unequal job size assumption is similar to MOJ scheduling where each order can have an individual size. Parsa *et al.* [30] propose a dynamic programming formulation, a branch-and-bound algorithm, and several constructive heuristics for the problem studied by Li *et al.* [18].

Overall, it turns out that to the best of our knowledge MOJ scheduling problems with a nonrestrictive common due date are not discussed so far in the literature. It seems reasonable to extend the related work for p-batching to the situation of MOJ scheduling problems.

3. Structural properties and computational complexity

We start by deriving structural properties of optimal solutions. Note that most of the properties carry over from the conventional scheduling setting where jobs with weights are considered and a weighted version of the E/T measure is used to the MOJ setting. The first property is valid for any optimal schedule for problem (2.3). In the remainder of this paper, we denote the processing time of job j by $\tilde{p}_j$.

Property 3.1. *In any optimal schedule, there does not exist any idle time between two consecutive jobs.*

Proof. Following Kanet [16], we assume that an optimal schedule S exists that contains idle time $\Delta > 0$ between the two consecutive jobs j and k with the property $e_j = e_k - \tilde{p}_k - \Delta < e_k - \tilde{p}_k \le d$. In this situation, job j contributes to the objective function value $E_j + T_j = |J_j|\,(d - e_j) = |J_j|\,(d - \{e_k - \tilde{p}_k - \Delta\})$. Here, we denote by $|J_j|$ the number of orders that are included in job j. By performing a right shift of job j we construct a schedule S^* with no idle time between job j and k. We obtain $E/T\,(S) - E/T\,(S^*) = |J_j|\,\Delta > 0$ which contradicts to the optimality of S. When idle time exists between jobs that complete after d then a left shift leads to the contradiction. □

The next property shows the existence of optimal schedules that do not contain straddling jobs.

Property 3.2. *There is an optimal schedule where a job is completed at the common due date.*

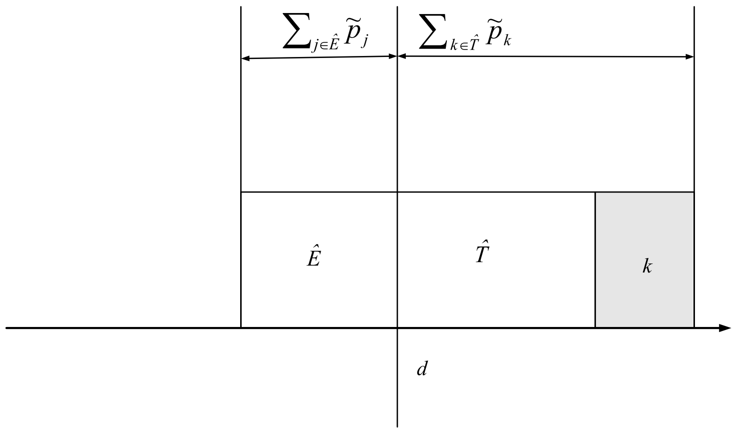
$\sum_{j\in\hat{E}}\tilde{p}_j$
$\sum_{k\in\hat{T}}\tilde{p}_k$
$\hat{E}$
$\hat{T}$
k
d

FIGURE 2. Situation in the proof of Property 3.4.

Proof. This proof is based on the technique used in the proof of Property 3.1 in Hall and Posner [12] where the number of orders in a job is interpreted as the weight for this job. This means that we assume that there is an optimal schedule where no job finishes at d. Using the same arguments as in the proof of Property 1 in Hall and Posner [12] we can show that sliding all jobs earlier or later until a job completes at d leads also to an optimal schedule. $\square$

The next property characterizes the sequence of jobs in a certain class of optimal schedules. Here, we denote the set of early jobs including the job with completion time d by $\hat{E}$ and the corresponding set of tardy jobs by $\hat{T}$. The related sets of orders are denoted by $\tilde{E}$ and $\tilde{T}$, respectively.

Property 3.3. *There is an optimal schedule where the jobs in $\hat{E}$ are sorted with respect to $|J_j|/\tilde{p}_j$ in non-decreasing order while the jobs in $\hat{T}$ are sorted with respect to $|J_j|/\tilde{p}_j$ in non-increasing order.*

Proof. We assume that there is an optimal schedule S where two jobs $j,k \in \hat{E}$ with $e_j < e_k$ exist such that $|J_j|/\tilde{p}_j > |J_k|/\tilde{p}_k$. Swapping the two jobs leads to an E/T change of $|J_k|\tilde{p}_j - |J_j|\tilde{p}_k < 0$. This contradicts with the optimality of S. The proof for jobs in $\hat{T}$ goes in an analogous way. $\square$

Note that Property 3.3 leads to a V-shaped schedule (*cf.* [12, 31]), *i.e.*, jobs before d have non-decreasing $|J_j|/\tilde{p}_j$ ratios, *i.e.*, they are sequenced according to the weighted longest processing time (WLPT) rule, whereas jobs after d have non-increasing $|J_j|/\tilde{p}_j$ ratios, *i.e.*, they are sequenced according to the weighted shortest processing time rule (WSPT).

The next property according to Hall and Posner [12] shows the distribution of jobs to the sets $\hat{E}$ and $\hat{T}$ in optimal schedules.

Property 3.4. *The sum of the processing times of the jobs that are in $\hat{E}$ is larger or equal to the sum of the processing times of the jobs in $\hat{T}$ in any optimal schedule.*

Proof. We assume that there is an optimal schedule S where $\sum_{j\in\hat{E}}\tilde{p}_j < \sum_{k\in\hat{T}}\tilde{p}_k$. Let k be the job that is scheduled at the last position in $\hat{T}$. This situation is shown in Figure 2.

Moving the job k at the first position in $\hat{E}$ leads to a schedule S^* with the change $|J_k|\left(\sum_{j\in\hat{E}}\tilde{p}_j - \sum_{k\in\hat{T}}\tilde{p}_k\right) < 0$ in the objective function value which contradicts with the optimality of S. Hence, our assumption about S is not correct. This means that we always have $\sum_{j\in\hat{E}}\tilde{p}_j \geq \sum_{k\in\hat{T}}\tilde{p}_k$ which is the property to be established. $\square$

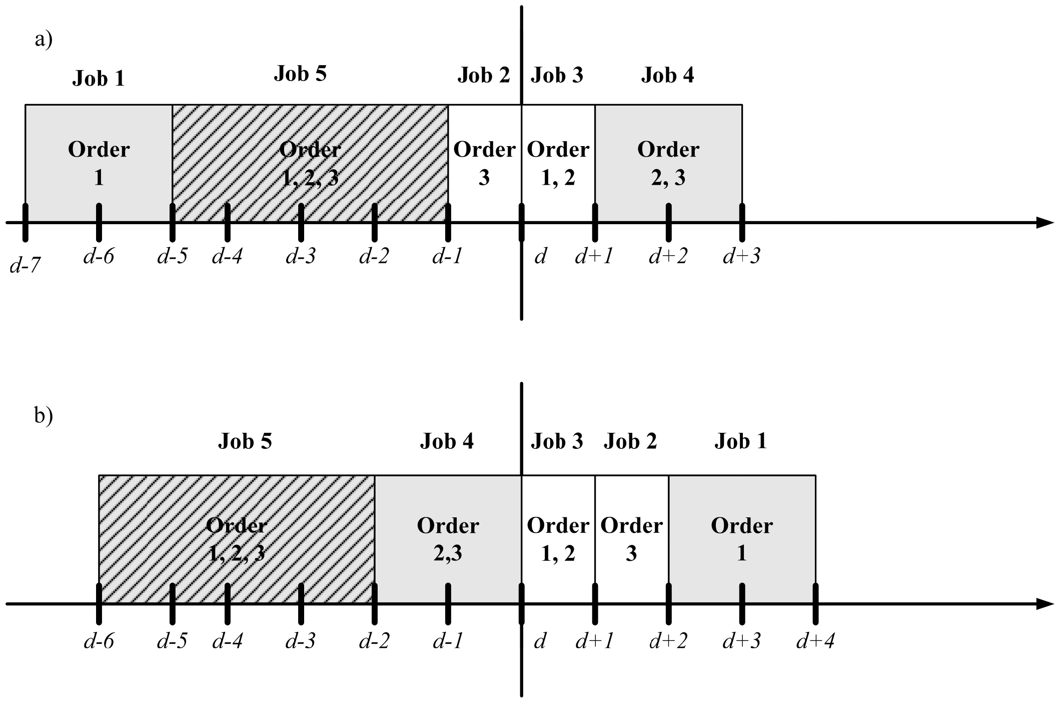

FIGURE 3. (a) Schedule that fulfills the four properties and (b) optimal schedule.

Note that the schedule shown in Figure 1 satisfies the Properties 3.1–3.3. This schedule can be turned into a schedule that fulfills also Property 3.4 by placing job 5 between job 1 and job 2 in $\hat{E}$. The corresponding optimal objective function value is $E/T = 16$. This schedule is shown in Figure 3a. An optimal schedule that is obtained by a different sequencing of the jobs from the previous schedule is depicted in Figure 3b. The optimal objective function value verified by the MILP (2.4)–(2.14) is $E/T = 14$.

Next, we consider the case of a single order family. In this situation, all jobs have an identical processing time $\tilde{p}_j \equiv p$. Using Properties 3.1, 3.2 and the algorithm SCHED proposed by Kanet [16] it is clear that the completion times of the jobs in an optimal schedule according to Hall and Posner [12] are given by:

$$e_j = d - \lceil F/2 \rceil + jp, \quad j = 1, \ldots, F. \tag{3.1}$$

The earliness–tardiness value associated with a single order of job j can be computed as follows:

$$E_j + T_j = p\left((\lceil F/2 \rceil - j)^+ + (j - \lceil F/2 \rceil)^+ \right). \tag{3.2}$$

Using expression (3.2), we are able to formulate the following IP that is simple compared to the MILP formulation (2.4)–(2.14):

$$\min \sum_{j=1}^{F} \sum_{o=1}^{N} (E_j + T_j)\, x_{oj} \tag{3.3}$$

subject to

$$\sum_{j=1}^{F} x_{oj} = 1,\ o = 1, \ldots, N, \tag{3.4}$$

$$\sum_{o=1}^{N} s_o x_{oj} \leq K,\ j = 1, \ldots, F, \tag{3.5}$$

$$x_{oj} \in \{0, 1\},\ o = 1, \ldots, N, j = 1, \ldots, F. \tag{3.6}$$

The IP model (3.3)–(3.6) is a special case of the minimization variant MINGAP of the generalized assignment problem (GAP). It is shown for GAP and MINGAP that even the feasibility question is NP-complete [19].

Next, we prefer to prove more directly that problem (2.3) is NP-hard, even when only a single family is considered. The proof is based on the strongly NP-hard 3-partition problem [7]. For the sake of completeness, we first recall this problem:

3-partition problem: Let B be a positive integer. Moreover, we assume that we have $3m$ integers $a_j, j = 1, \ldots, 3m$, such that$B/4 < a_j < B/2, j = 1, \ldots, 3m$ and $\sum_{j=1}^{3m} a_j = mB$. Do there exist m pairwise disjoint triples $A_i \subseteq \{1, \ldots, 3m\}$ such that $\sum_{j \in A_i} a_j = B$ for $i = 1, \ldots, m$?

The 3-partition problem asks whether there exists a solution for a given instance of the 3-partition problem or not.

Theorem 3.5. *Problem* (2.3) *is NP-hard in the strong sense even in the case that all orders belong to a single family.*

Proof. We start by considering a specific instance of problem (2.3) for the case of a single family given by $N := 3m$, $F := m$, $s_j := a_j$, $K := B$,$p := 1$. The threshold value of the E/T measure is defined by $Y := 3\sum_{j=1}^{F}\left((\lceil F/2 \rceil - j)^+ + (j - \lceil F/2 \rceil)^+\right)$. The decision version of the scheduling problem $1|moj\,(lot)\,, C_o \equiv d|E/T$ asks whether there is a feasible schedule S such that $E/T\,(S) \leq Y$ holds or not. We shall show that a feasible solution of this instance with objective function value smaller or equal to $3\sum_{j=1}^{F}\left((\lceil F/2 \rceil - j)^+ + (j - \lceil F/2 \rceil)^+\right)$ exists if and only if a solution of 3-partition exists for the given input data.

We assume first that there is a solution of 3-partion. We then have sets $A_i, i = 1, \ldots, m$ such that $\sum_{j \in A_i} s_j = K, i = 1, \ldots, m$, *i.e.*, the sets $A_i, i = 1, \ldots, m$ determine the content of the jobs. Each order is assigned to exactly one of the A_i sets since the A_i are disjoint, and we have $\sum_{i=1}^{3m} s_i = mK$. The objective function value of the resulting schedule is clearly $3\sum_{j=1}^{F}\left((\lceil F/2 \rceil - j)^+ + (j - \lceil F/2 \rceil)^+\right)$.

Next, we assume that there is a feasible solution of the instance of the scheduling problem with objective function value smaller than or equal to $3\sum_{j=1}^{F}\left((\lceil F/2 \rceil - j)^+ + (j - \lceil F/2 \rceil)^+\right)$. We assume first that more than three orders belong to a single job. This then contradicts to the conditions $K/4 < s_o, o = 1, \ldots, N$ and the fact that the capacity of each job is K. If less than three orders are in a job j then we have $\sum_{o \in j} s_o < K$ due to $s_o < K/2, o = 1, \ldots, N$. This leads to $\sum_{o=1}^{N} s_o < FK$ which contradicts to $\sum_{o=1}^{N} s_o = FK$. Overall, we have F jobs, each of them contains exactly three orders where the sum of the order sizes is exactly K for each job. This is a solution of 3-partition. □

Because of the shown NP-hardness of problem (2.3), we have to propose efficient heuristics, especially metaheuristics, to solve large-sized problem instances in a reasonable amount of computing time. It is desirable to apply the derived properties of optimal schedules when metaheuristics are designed. This will lead to hybrid algorithms.

4. Heuristic and exact solution approaches

We start by describing simple reference heuristics in Section 4.1. We then present several heuristics based on GAs in Sections 4.2 and 4.3. An IP-based exact algorithm is proposed for the special case of a single family in Section 4.4.

4.1. Simple reference heuristics

We discuss strategies to form jobs and sequence them. Therefore, we use bin packing-type algorithms that are inspired by Mason *et al.* [21]. First, the orders have to be sorted. Since we have equal weights for all orders, we can only apply the smallest size (SS) and the largest size (LS) rule. The first rule sorts the orders with respect to the index $I(o) := s_o$ in non-decreasing order, while the second one applies this index in non-increasing order.

Next, we describe assignment strategies for a given sorted list of orders. These strategies are called order-to-job-assignment rules in Mason *et al.* [21]. Among the different assignment strategies in Mason *et al.* [21] especially the FFDn and the FFDAJS strategies including the FFD1 strategy are successful in some preliminary experimentation. The remaining strategies in Mason *et al.* [21], namely FFD1b, FFDnb, and FFD-AJS1, use all the available FOUPs. This is clearly an undesirable feature for MOJ problems with lot processing. Therefore, we recall only the FFDn, FFD1, and FFDAJS strategies in this subsection. We assume without loss of generality that we have a fixed sequence of the jobs. The three strategies are as follows:

1. **FFD1:** This rule assigns orders from a sorted list until no more orders from the list can be placed in the current job due to the capacity constraints. In this situation, the next job is considered and the procedure is repeated. Note that this approach might lead to F passes through the order list.
2. **FFDn:** This strategy considers again orders from the list. It assigns the current order to the first job with enough capacity to place the order. If not enough capacity is available in the currently considered job, the next job is used. It is clear that this strategy leads only to a single pass through the order list.
3. **FFDAJS:** The goal of this assignment strategy consists in assigning orders to the jobs in such a way that the capacity usage of all jobs is similar. Therefore, the sum of the sizes of all unassigned orders will be divided by the number of empty jobs. This provides the expected average job size for each remaining job. The next job is then filled using the FFD1 strategy without exceeding the expected average job size and violating any of the constraints.

Note that the order-to-job assignment rule based on the FFDAJS strategy requires knowing how many empty jobs of each incompatible family still exist. Therefore, we use the expression

$$F_l := \lfloor F/L \rfloor, \quad l = 1, \ldots, L \tag{4.1}$$

for determining the number of jobs F_l that belong to family l. If there are still jobs left they will be assigned to the families if needed. In the case of the remaining order-to-job assignment rules a family is assigned to a job whenever a job is needed for orders of the corresponding family.

Next, we sketch an algorithm that assigns the already formed jobs to the early and tardy order/job sets. Let be J the set of all jobs. We introduce job sets for partial schedules of early and tardy jobs by the definitions $\bar{E} := \left\{ j \in J | j \text{ is part of the partial schedule for } \hat{E} \right\}$ and $\bar{T} := \left\{ j \in J | j \text{ is part of the partial schedule for } \hat{T} \right\}$. The early-tardy assignment heuristic works as follows:

Early-Tardy Assignment Heuristic (ETAH)

1. Sort the jobs according to the index $I_j := |J_j| / \tilde{p}_j$ in non-increasing order. Initialize $\bar{E} := \emptyset$ and $\bar{T} := \emptyset$.
2. Compute the sum of the processing times of the already sequenced jobs in $\hat{E}$ and $\hat{T}$, namely $\sum_{j \in \bar{E}} \tilde{p}_j$ and $\sum_{j \in \bar{T}} \tilde{p}_j$. Select the first job $j^* \in J$. If $\sum_{j \in \bar{E}} \tilde{p}_j < \sum_{j \in \bar{T}} \tilde{p}_j + \tilde{p}_{j^*}$ then go to Step 3, otherwise go to Step 4.
3. Insert job j^* as first job in $\hat{E}$. Update $\bar{E} := \bar{E} \cup \{j^*\}$.

4. Insert job j^* as last job in $\hat{T}$. Update $\bar{T} := \bar{T} \cup \{j^*\}$.
5. Update $J := J \backslash \{j^*\}$. If $J \neq \emptyset$ then stop, otherwise go to Step 2.

Step 1 and Steps 3 and 4 ensure that Property 3.3 is respected by the resulting schedule. Straddling jobs are avoided by Steps 3 and 4. Property 3.4 is taken into account by Steps 2, 3, and 4. Note that the quality of a schedule resulting from the ETAH procedure cannot be improved by exchanging the positions of jobs in the schedule. If we use FFDn and SS for job formation and ETAH for sequencing, we call the resulting heuristic ETAH(FFDn-SS). The abbreviations ETAH(FFDn-LS), ETAH(FFDAJS-SS), and ETAH(FFDAJS-LS) are used with a similar meaning. We suppress the term ETAH in the names of the simple heuristics at some places.

4.2. Genetic algorithms based on early-tardy set decomposition

GAs are widely used approaches to solve large-scale combinatorial optimization problems (*cf.* [8, 22]). GAs represent an important class of population-based metaheuristics. The individuals of the population are called chromosomes. A fitness value derived from the objective function value of the scheduling problem is assigned to each chromosome. GAs work iteratively, *i.e.*, a generation corresponds to a single iteration. The individuals of the new generation are obtained from the individuals of the previous one by applying reproduction, *i.e.* crossover and selection, and mutation. It is more likely that individuals with a good fitness value are selected for the next generation.

The first GA to solve problem (2.3) is used to determine the sets $\tilde{E}$ and $\tilde{T}$. A similar technique is applied in Mönch *et al.* [24]. An order-based representation is used. Therefore, the chromosomes are arrays of length N, *i.e.*, we obtain

$$c := (a_1, \ldots, a_N), \tag{4.2}$$

where we set for $o = 1, \ldots, N$

$$a_o := \begin{cases} 1, & \text{if } o \in \tilde{E} \\ 0, & \text{otherwise.} \end{cases} \tag{4.3}$$

We consider the following chromosome for $N = 6$ orders as an example:

$$c := (1,\, 0,\, 0,\, 1,\, 1,\, 0). \tag{4.4}$$

The chromosome (4.4) leads to the following set $\tilde{E} := \{1, 4, 5\}$ of early and on-time orders, whereas the set of tardy orders is $\tilde{T} := \{2, 3, 6\}$.

In order to calculate the fitness of a chromosome it is first decoded into a schedule by applying any of the order sorting and job formation strategies described in Section 4.1. Here, we use the SS and LS strategies to sort the orders and the FFDn procedure for order-to-job assignments. Once the jobs in both $\hat{E}$ and $\hat{T}$ are formed they are sequenced using Property 3.3. When a chromosome leads to an infeasible solution than the objective function value will be penalized.

The initial population is randomly generated. A standard one-point crossover [22] is used. Therefore, two parent chromosomes $c_1 := (a_1, \ldots, a_N)$ and $c_2 := (b_1, \ldots, b_N)$ are chosen from the population by applying the roulette wheel procedure (see [22]). A cutting position $1 \leq l \leq N$ is randomly chosen. The two child chromosomes $c_3 := (a_1, \ldots, a_{l-1}, b_l, \ldots, b_N)$ and $c_4 := (b_1, \ldots, b_{l-1}, a_l, \ldots, a_N)$ are obtained by exchanging the two substrings starting at position l.

Moreover, flip mutation is used, *i.e.*, we choose randomly a gene a_k. This gene is then replaced by $a_k := 1 - a_k$, *i.e.*, the assignment of order k is changed from the set $\hat{E}$ to $\hat{T}$ and vice versa. Mutation is applied to a specific chromosome with a mutation probability p_M. Once a chromosome is chosen for mutation, the probability p_M determines which genes are mutated.

A steady-state GA with overlapping populations is used, *i.e.*, the individuals of the population with the lowest fitness are replaced. It is known from Syswerda [38] and Rogers and Prügel-Bennett [33] that this class of GAs is computationally efficient. The amount of chromosomes used for recombination purposes is determined by the crossover probability p_c. The replacement probability is denoted by p_r. The proposed GAs are abbreviated by GA-FFDn-SS and GA-FFDn-LS, respectively.

4.3. Order sequence-based genetic algorithm

While simple order sequencing strategies are applied in ETAH-type heuristics (*cf.* Sect. 4.1), the second GA proposes order sequences similar to Sobeyko and Mönch [36] for another MOJ scheduling problem (see Sect. 2.3) that will be applied to form jobs using the methods described in Section 4.1. Therefore, much more order sequences are considered compared to the simple order sequencing strategies. However, the decoding scheme in this research is more difficult than the one used by Sobeyko and Mönch [36].

Since it is well-known that RKGAs are appropriate for sequencing-type problems (*cf.* [2, 9]) we apply the random-key representation for encoding order permutations. We use again an order-based representation that leads to an array of length N as shown in (4.2), *i.e.*, we obtain:

$$c := (z_1, \ldots, z_N)\,, \tag{4.5}$$

where a realization z_o of a random variable $Z \sim U\,[0,1]$ is assigned to position o within the chromosome in the initial population. Here, we denote by $U\,[a,b]$ a continuous uniform distribution over the interval $[a,b]$. The z_o values are called random keys. The orders are sorted in non-decreasing order of the random keys. The following example for $N=6$ is considered. We have the chromosome:

$$c := (0.80,\ 0.50,\ 0.90,\ 0.11, 0.23, 0.64)\,. \tag{4.6}$$

This chromosome is decoded into the following order sequence:

$$4 \to 5 \to 2 \to 6 \to 1 \to 3, \tag{4.7}$$

where job 4 is on the first position and job 3 on the last position.

The genetic operators work on a population of random-key vectors. An elitist strategy to carry out the reproduction of the population, *i.e.*, the best individuals are copied from one generation to the next one, is applied. A parameterized uniform crossover is used, *i.e.*, two parents are randomly selected from the population by the roulette wheel procedure. A biased coin is then tossed for each gene to determine which parent will contribute to the allele. Immigration as proposed by Bean [2] is used to introduce some diversity into the population, *i.e.*, at each generation, a certain percentage of new individuals from the population are generated randomly in a similar manner as for the initial population.

In order to determine the fitness of each chromosome, the ETAH procedure described in Section 4.1 in combination with the FFDn approach serves as a decoder that generates a feasible schedule from the chromosome. This means that jobs are formed by FFDn based on the order sequence obtained from the chromosome, and the ETAH procedure assigns the jobs to the $\hat{E}$ and the $\hat{T}$ sets and sequences them in each of the two sets. When a chromosome leads to an infeasible solution then a penalty term is used in the objective function. The resulting GA is called RKGA in the remainder of this paper.

We summarize and compare the different hybridization strategies in the GA-FFDn-type approaches and in the RKGA scheme in Table 1.

Note that other hybridization strategies are possible. One of them is sketched in Section 6. We can see from Table 1 that the proposed hybridization strategies rely on the properties derived in Section 3. Although the

Table 1. Hybridization strategies in the two proposed GAs.

	GA	RKGA
Encoding scheme	Array of orders, each gen $c_o \in \{0,1\}$	Array of orders, each $c_o \in [0,1]$
Order sequencing	SS/LS	Taken from a chromosome
Job formation	Based on FFDn, separate for $\tilde{E}, \tilde{T}$	FFDn for $\hat{E} \cup \hat{T}$
Job sequencing	Based on the LPT rule for $\hat{E}$ and based on the SPT rule for $\hat{T}$	Based on the ETAH scheme for $\hat{E} \cup \hat{T}$

encoding schemes of the two GAs are taken from Mönch *et al.* [24] and Sobeyko and Mönch [36], the algorithms in the present paper are different from these GA-type heuristics in the two previous papers because of the problem-specific hybridization approaches.

4.4. Solution approaches for the special case of a single family

In the special case of a single family, we solve the IP (3.3)–(3.6) by a commercial solver. This leads to a fast approach that provides the exact solution in the case of small- or even medium-sized problem instances. If the instances are large, we solve the IP (3.3)–(3.6) in a heuristic manner, *i.e.*, we do not solve the IP to optimality, determining a feasible solution within a given amount of computing time is enough.

Moreover, the simple heuristics from Section 4.1 and the two GA-based approaches from Sections 4.2 and 4.3 are applied to solve instances of problem (2.3) in the single family case. It is especially interesting to apply the exact approach for medium-sized problem instances since we can assess the performance based on the known optimal solutions that are computed using the IP approach.

5. Computational experiments

In Section 5.1, the design of experiments is described while parameter setting and implementation issues are discussed in Section 5.2. The results of computational experiments are presented in Section 5.3. The results obtained are analyzed and discussed in Section 5.4.

5.1. Design of experiments

We expect that the performance of the different heuristics depends on the number of orders, the number of incompatible families, and the capacity of the FOUPs. Therefore, we randomly generate problem instances for these factors. The number of FOUPs is given by the expression $\lceil N\nu/12\beta \rceil + L$ where the factor ν determines the size of the orders and the factor β is responsible for sizing the capacity of the FOUPs. The processing time of the jobs that contain orders of a given family are taken from a given discrete distribution. They are individually chosen for the families of each problem instance. The obtained design of experiments is summarized in Table 2.

Overall, we consider 640 small-, medium- and large-sized problem instances in the computational experiments. Although we are unable relying our experiments on real-world problem instances, our instance generation scheme is motivated by process conditions, for instance order size or processing time distributions, found in real-world wafer fabs. The maximum amount of 120 and 240 orders for a single machine reflects the load situation in a heavily congested wafer fab.

Moreover, we perform additional computational experiments with small-sized problem instances according to the design of experiments described in Table 3. In total, 72 small-sized problem instances are solved.

In a final experiment, we are interested in assessing the performance of the GA-type approaches based on the IP approach (3.3)–(3.6). Therefore, we generate 160 additional single-family problem instances based on the design summarized in Table 4.

Table 2. Design of experiments for small-, medium- and large-sized problem instances.

Factor	Level	Count
N	$30, 60, 120, 240$	4
s_o	$\sim DU\left[\nu - \frac{v+1}{2}, \nu + \frac{\nu+1}{2}\right],\ \nu \in \{3,5\}$	2
K (in wafers)	$12\beta + 1,\ \beta \in \{1,2\}$	2
L	$1, 3, 6, 10$	4
Processing time distribution	5 with $p = 0.2$	1
	4 with $p = 0.2$	
	10 with $p = 0.3$	
	16 with $p = 0.2$	
	20 with $p = 0.1$	
Number of independent replications	10 per factor combination	10
Total number of problem instances		640

Table 3. Design of experiments for small-sized instances with incompatible families.

Factor	Level	Count
N	$12, 24, 36$	3
s_o	$\sim DU\left[\nu - \frac{v+1}{2}, \nu + \frac{\nu+1}{2}\right],\ \nu \in \{3,5\}$	2
K (in wafers)	$12\beta + 1,\ \beta \in \{1,2\}$	2
L	$3, 6$	2
Processing time distribution	5 with $p = 0.2$	1
	4 with $p = 0.2$	
	10 with $p = 0.3$	
	16 with $p = 0.2$	
	20 with $p = 0.1$	
Number of independent replications	3 per factor combination	3
Total number of problem instances		72

Table 4. Design of experiments for single-family instances.

Factor	Level	Count
N	$30, 60, 120, 240$	4
s_o	$\sim DU\left[\nu - \frac{v+1}{2}, \nu + \frac{\nu+1}{2}\right],\ \nu \in \{3,5\}$	2
K (in wafers)	$12\beta + 1,\ \beta \in \{1,2\}$	2
Processing time	10	1
Number of independent replications	10 per factor combination	10
Total number of problem instances		160

We are interested in the E/T value obtained by a specific heuristic H relative to the E/T value computed by the ETAH(FFDn-LS) approach, *i.e.*, we determine the performance ratio:

$$PR\,(H) := \frac{E/T\,(H)}{E/T\,(ETAH\,(FFDn - LS))} \tag{5.1}$$

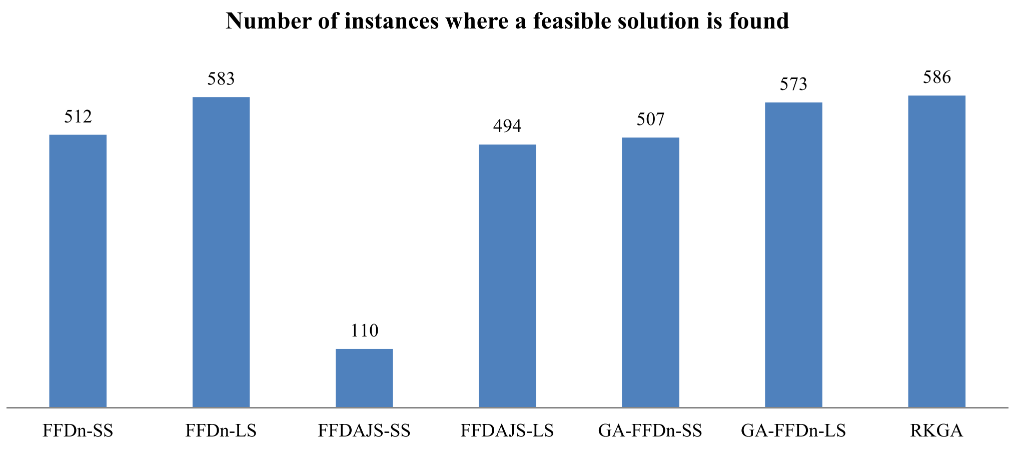

FIGURE 4. Number of instances where a feasible solution is found by the corresponding heuristic.

for each instance. The ETAH(FFDn-LS) scheme is used for comparison since this simple heuristic is able to provide feasible solutions for a large number of problem instances from Table 2 as shown in Figure 4.

Moreover, we know for small-sized problem instances the optimal (or near-to-optimal) objective function values from the MIP (2.4)–(2.14) or the IP (3.3)–(3.6) in the single-family case. Therefore, in this situation, we replace the ratio (5.1) by the ratio

$$RE\,(H) := \frac{E/T\,(H)}{E/T\,(MIP)} \tag{5.2}$$

that shows the E/T values of the heuristics relative to the E/T values of exact solutions. All heuristics with stochastic ingredients are replicated ten times with different seeds to obtain statistically reasonable results. The average value of the corresponding ratios (5.1) and (5.2) is reported over all replications that result in a feasible schedule. A computing time of 10 s is used per problem instance for the three GAs to allow a fair comparison. Moreover, we also conduct a second series of experiments where we allow 20 s of computing time per problem instance.

5.2. Parameter settings and implementation issues

A population size of 300 is used for the GA-FFDn-SS and the GA-FFDn-LS approach, respectively. The mutation probability is chosen as $p_M = 0.1$. The corresponding crossover probability is selected as $p_c = 0.8$, while the replacement probability is set to $p_R = 0.9$. These parameter settings are determined based on preliminary experiments with 16 randomly selected problem instances out of the 640 instances based on Table 2. For the population size, the tested values are $\{300, 500, 700\}$. Moreover, we test the settings $p_M \in \{0.10, 0.15, 0.20\}$, $p_c \in \{0.80, 0.85, 0.90\}$, and $p_R \in \{0.80, 0.90\}$. We consider parameter combinations according to a Taguchi Orthogonal Array and apply the corresponding GA variant to all 16 instances to obtain the parameter settings shown above.

For the RKGA, we apply again a population size of 300. This setting is chosen based on a trial and error strategy based on the population sizes from the set $\{300, 500, 700\}$ and the 16 instances described before.

The remaining parameters for the RKGA scheme are directly taken from Bean [2], *i.e.*, an immigration rate of 1% is used, 20% of the chromosomes with the best fitness values are copied into the next generation, and the parameterized uniform crossover is applied to the entire population with the probability of tossing a head that is 0.7. For the instances in Tables 2 and 3, a maximum computing time of 2 h per instance is used for a commercial solver. The proposed GAs are performed for 300 generations.

TABLE 5. Computational results for the ETAH procedure.

FFDn-SS	FFDn-LS	FFD1-SS	FFD1-LS	FFDAJS-SS	FFDAJS-LS
0.930	1.000	0.929	1.001	1.314	1.650

The proposed algorithms are coded using the C++ programming language. Moreover, the GAs are implemented based on the GaLib framework [41]. The presented chromosome representation for the RKGA is coded based on the class GARealGenome of the GALib framework, whereas the class GA1DArrayGenome<int> is applied for the second type of proposed GA. The MIP (2.4)–(2.14) and the IP (3.3)–(3.6) are solved using the commercial solver CPLEX 12.1. The experiments with the commercial solver are performed on an Intel Core i7-2600 CPU 3.40 GHz computer with 16 GB RAM whereas the remaining computational experiments are conducted on an Intel Core i7-3632QM CPU 2.20 GHz computer with 8 GB RAM.

5.3. Computational results

In a first experiment, we are interested in the performance of the ETAH heuristic with respect to the applied order sorting and job formation strategies from Section 4.1. We consider the 640 instances generated according to the design in Table 2. The corresponding average PR values over all instances are presented in Table 5.

Since the FFDn and FFD1order-to-job assignment procedures perform very similar, we will consider only FFDn in the remaining experiments. Moreover, we show only results for the ETAH(FFDn-SS) and ETAH(FFDn-LS) schemes since the ETAH(FFDn-LS) heuristic outperforms both the ETAH(FFAJS-SS) and the ETAH(FFAJS-LS) scheme to a large extent.

We start by presenting the results for computational experiments with small- and medium-sized problem instances and a computing time of 10 second per problem instance. The computational results are shown in Table 6. Best results in a row are marked bold.

Instead of comparing the problem instances individually, the instances are grouped according to factor levels such as number of incompatible families, order size (ν), and FOUP capacity (β). Therefore, 10 instances are considered in each row. The corresponding number of instances with feasible solutions obtained from ETAH(FFDn-LS) is shown in the last column of Table 4. If a heuristic is unable to find feasible solutions we indicate this by the n/a symbol. The E/T values of the heuristics are shown relative to the corresponding ETAH(FFDn-LS) value, *i.e.*, we present the average of the ratios $PR(H)$ over all instances with feasible solutions.

We provide the results for large-sized problem instances in a similar way in Table 7.

Computational results for the 320 large-sized problem instances of Table 2 and a computing time of 20 s per instance is shown in Figure 5 together with the corresponding results for 10 s of computing time. We do not present the results for the small- and medium-sized instances of Table 2 since the difference to the results obtained by applying 10 s per instance are much smaller.

In a second series of experiments, we are interested in assessing the quality of the proposed heuristics based on the 72 small-sized problem instances found in Table 3. We present only results for the instances with less than 36 orders since for most of the instances with $N = 36$ even after 2 h of computing time the solver was not able to compute a feasible solution. Here, the ratio $RE(H)$ is shown in Table 8. For a single instance with $N = 24$ the solver was not able to find a feasible solution. This instance is excluded from Table 8. Overall, 47 instances are considered in this table. Three instances are considered in each row with the exception that a single row is based on two instances. Again, best results for the heuristics in a given row are marked bold. We also show the relative MIP gap, defined as the ratio of the difference of the best integer solution and the best lower bound and the best integer solution, in percent.

Computational experiments with the IP formulation (3.3)–(3.6) are also performed using CPLEX to assess the solution quality in the special case of a single family. It turns out that for small-sized instances, *i.e.* for $N \leq 60$, optimal solutions are found by CPLEX on average within less than 20 s per instance. The RKGA is

Table 6. Computational results for small- and medium-sized problem instances.

L	ν	β	FFDn-SS	FFDn-LS	GA-FFDn-SS	GA-FFDn-LS	RKGA	# instances
					$N = 30$			
1	3	1	0.889	1.000	0.862	0.887	**0.856**	10
	3	2	0.814	1.000	**0.809**	**0.809**	**0.809**	10
	5	1	n/a	1.000	n/a	0.948	**0.919**	10
	5	2	0.932	1.000	0.891	0.905	**0.883**	10
3	3	1	0.885	1.000	**0.792**	0.800	0.827	10
	3	2	0.914	1.000	**0.721**	0.727	0.867	10
	5	1	n/a	1.000	0.827	**0.771**	0.795	8
	5	2	0.925	1.000	**0.716**	0.741	0.788	10
6	3	1	0.997	1.000	0.812	**0.809**	0.921	10
	3	2	1.000	1.000	0.759	**0.752**	0.860	10
	5	1	1.007	1.000	0.819	**0.790**	0.846	10
	5	2	1.003	1.000	0.838	**0.832**	0.921	10
10	3	1	1.000	1.000	0.818	**0.797**	0.881	10
	3	2	1.000	1.000	**0.751**	0.756	0.844	10
	5	1	1.000	1.000	0.839	**0.827**	0.917	10
	5	2	1.000	1.000	0.743	**0.735**	0.844	10
Overall			0.955	1.000	**0.800**	0.805	0.861	
					$N = 60$			
1	3	1	0.902	1.000	0.880	0.921	**0.872**	10
	3	2	0.895	1.000	**0.885**	0.912	**0.885**	10
	5	1	n/a	1.000	n/a	0.968	**0.943**	9
	5	2	0.995	1.000	0.965	**0.961**	0.968	10
3	3	1	0.856	1.000	**0.778**	0.785	0.787	10
	3	2	0.904	1.000	**0.790**	0.813	0.841	10
	5	1	0.969	1.000	0.885	0.846	**0.837**	7
	5	2	0.931	1.000	0.845	**0.829**	0.835	10
6	3	1	0.921	1.000	**0.798**	0.803	0.822	10
	3	2	0.985	1.000	**0.830**	**0.830**	0.891	10
	5	1	n/a	1.000	n/a	**0.766**	0.796	7
	5	2	0.943	1.000	0.802	**0.794**	0.811	10
10	3	1	0.937	1.000	**0.827**	0.830	0.888	10
	3	2	1.000	1.000	0.782	**0.773**	0.847	10
	5	1	0.986	1.000	0.846	**0.811**	0.838	10
	5	2	0.895	1.000	**0.792**	0.801	0.854	10
Overall			0.937	1.000	**0.836**	0.840	0.857	

able to determine these optimal solutions for the majority of the problem instances. Starting with $N = 120$ there are instances that require much more computing time to prove the optimality of a solution, often even 2 h of computing time are not enough. For some of those instances, the RKGA is able to find slightly better solutions. Due to space limitations, we do not show the detailed results.

5.4. Analysis and discussion of the results

We see from the Tables 6 and 7 that the RKGA perform well under most of the experimental conditions. The RKGA is slightly outperformed by the GA-FFDn-SS scheme when the FOUP capacity is 25 wafers and when instances with a single family are considered for $N = 240$. In the first situation, the grouping decisions are more

Table 7. Computational results for large-sized problem instances.

L	ν	β	FFDn-SS	FFDn-LS	GA- FFDn-SS	GA-FFDn-LS	RKGA	# instances
					$N = 120$			
1	3	1	0.918	1.000	**0.906**	0.954	0.909	10
	3	2	0.917	1.000	**0.904**	0.942	0.933	10
	5	1	n/a	1.000	n/a	0.986	**0.967**	10
	5	2	n/a	1.000	n/a	**0.990**	1.013	10
3	3	1	0.896	1.000	0.849	0.847	**0.816**	10
	3	2	0.906	1.000	0.857	0.875	**0.847**	10
	5	1	n/a	1.000	n/a	0.848	**0.836**	2
	5	2	n/a	1.000	0.896	0.904	**0.843**	9
6	3	1	0.873	1.000	0.810	0.829	**0.787**	10
	3	2	0.895	1.000	0.844	0.869	**0.839**	10
	5	1	n/a	1.000	0.938	0.914	**0.860**	4
	5	2	0.920	1.000	0.857	0.869	**0.817**	10
10	3	1	0.860	1.000	0.805	0.825	**0.779**	10
	3	2	0.861	1.000	0.809	0.820	**0.773**	10
	5	1	n/a	1.000	0.917	0.871	**0.828**	4
	5	2	0.858	1.000	0.815	0.850	**0.793**	10
Overall			0.891	1.000	0.862	0.887	**0.853**	
					$N = 240$			
1	3	1	0.922	1.000	**0.917**	0.986	0.951	10
	3	2	0.946	1.000	**0.941**	0.985	1.005	10
	5	1	n/a	1.000	n/a	**0.993**	1.002	10
	5	2	n/a	1.000	n/a	**0.991**	1.046	10
3	3	1	0.912	1.000	0.872	0.902	**0.864**	10
	3	2	0.918	1.000	0.873	0.936	**0.868**	10
	5	1	n/a	1.000	n/a	0.898	**0.874**	4
	5	2	0.995	1.000	0.958	0.962	**0.936**	8
6	3	1	0.896	1.000	0.859	0.919	**0.849**	10
	3	2	0.907	1.000	0.879	0.941	**0.855**	10
	5	1	n/a	n/a	n/a	n/a	n/a	0
	5	2	n/a	1.000	0.932	0.966	**0.893**	10
10	3	1	0.875	1.000	0.861	0.928	**0.824**	10
	3	2	0.809	1.000	0.797	0.882	**0.759**	10
	5	1	n/a	1.000	n/a	0.944	**0.918**	1
	5	2	0.948	1.000	0.908	0.960	**0.851**	10
Overall			0.913	1.000	**0.891**	0.946	0.900	

important, but because the RKGA is based on the ETAH(FFDn) procedure some of the early decisions are often not effective. But these decisions are irrevocable. The scheduling problem becomes harder when more than one incompatible family is included in the problem instance. In the case of large-sized instances, *i.e.* $N = 240$ and only one family the search space of the RKGA scheme is huge since a permutation representation is used. This effect is less important for the GA-FFDn-SS scheme that uses the GA only to make assignment decisions for orders.

The difference between the performance of the RKGA scheme and the GA-FFDn-type algorithms is smaller when instances with $N \leq 60$ are considered. In this situation, even GA-FFDn-LS performs fairly well. We see from Table 6 that on average the GA-FFDn-SS scheme slightly outperforms the other two GAs. However, the

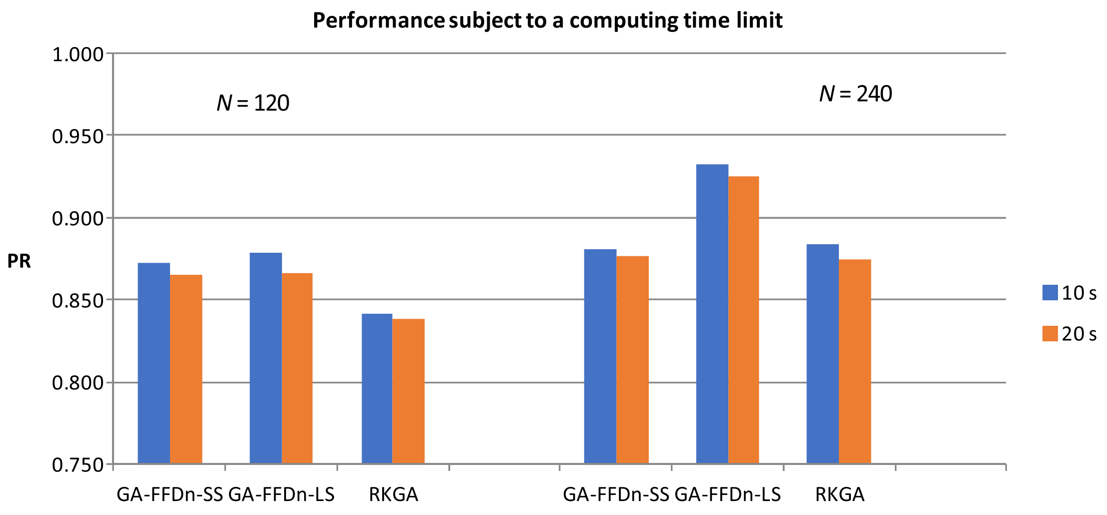

FIGURE 5. Overall improvements when 20 s of computing time are used.

TABLE 8. Computational results for small-sized problem instances.

	L	ν	β	FFDn-SS	FFDn-LS	GA-FFDn-SS	GA-FFDn-LS	RKGA	MIP-Gap (%)
						$N = 12$			
1	3	3	1	1.560	1.282	**1.000**	1.167	1.060	0
2		3	2	1.878	1.593	**1.000**	**1.000**	**1.000**	0
3		5	1	1.264	1.346	1.087	1.117	**1.046**	0
4		5	2	1.051	1.167	1.083	1.045	**1.000**	0
1	6	3	1	1.334	1.282	1.082	1.073	**1.000**	0
2		3	2	1.541	1.639	1.108	1.018	**1.000**	0
3		5	1	1.184	1.240	1.106	1.043	**1.000**	85
4		5	2	1.585	1.396	1.014	1.037	**1.000**	57
	Overall			1.424	1.368	1.060	1.062	**1.013**	18
						$N = 24$			
1	3	3	1	1.520	1.686	**1.076**	1.264	1.343	100
2		3	2	1.405	1.524	**1.000**	1.052	**1.000**	0
3		5	1	1.247	1.299	1.076	**1.034**	1.097	100
4		5	2	1.230	1.493	**1.031**	1.085	1.156	86
1	6	3	1	1.379	1.583	1.067	1.087	**1.037**	98
2		3	2	1.281	1.700	1.067	1.021	**1.000**	90
3		5	1	1.331	1.310	1.061	1.037	**1.035**	100
4		5	2	1.563	1.442	1.028	**1.000**	**1.000**	100
	Overall			1.370	1.505	**1.051**	1.073	1.084	84

value of the average values is limited due to the fact that GA-FFDn-SS is often unable to determine feasible solutions.

Among the simple heuristics ETAH(FFDn-LS) provides a feasible solution for many instances (see Fig. 4). However, when the ETAH(FFDn-SS) provides a feasible solution it often outperforms the ETAH(FFDn-LS) scheme as shown in Table 5. ETAH(FFDn-LS) outperforms ETAH(FFAJS-SS) and ETAH(FFAJS-LS) (see Tab. 5). It is interesting to observe from Figure 4 that the RKGA is able to find feasible solutions for a large

Table 9. Results of the Wilcoxon signed-rank test with a 1% significance level.

	FFDn-LS	GA-FFDn-LS	FFDn-SS	RKGA	GA-FFDn-SS
FFDn-LS		>	>	>	>
GA-FFDn-LS			>	>	>
FFDn-SS				>	>
RKGA					>

portion of the instances. This is caused by the permutation representation that is able to avoid infeasible solutions.

A doubling of the computing time per instance from 10 to 20 s leads only to minor performance improvements as shown in Figure 5. This is true for all hybrid algorithms.

The analysis of the results in Table 8 shows that the results obtained by the three GAs are pretty close to the MIP results obtained after 2 h of computing time. For most of the 47 considered problem instances the RKGA scheme provides the best solutions, followed by GA-FFDn-SS. This indicates that the GAs are correctly coded. We also see that even for $N = 24$ the MIP gap is large.

The results obtained for the single-family instances described in Table 4 demonstrate that up to 60 orders there is no need to use metaheuristics. When instances with more than 60 orders are considered it turns out that the RKGA is a good choice to obtain near-to-optimal solutions.

The Wilcoxon signed-rank test [42] with a significance level of 1% is applied to show the performance differences of the proposed heuristics based on the 640 problem instances from Table 2. Table 9 shows the obtained results. A ">" sign indicates that the heuristic of the column performs significantly better than the algorithm of the row.

The table demonstrates that the GA-FFDn-SS is the best performing algorithm followed by the RKGA scheme.

6. Conclusions and future research

In this paper, we discussed a single-machine MOJ for a lot processing environment with a nonrestrictive common due date. The E/T performance measure is considered. Only orders that belong to the same order family can be grouped together into one job. We derived several structural properties for optimal solutions of this scheduling problem. A MILP formulation was proposed. Moreover, we proved that this scheduling problem is NP-hard even in the situation that all orders belong to a single family.

We proposed several simple heuristics. Moreover, GAs that assign orders to the set of early and tardy jobs, respectively are designed. We presented also a RKGA that is based on the idea to sequence orders and then assign these orders to jobs. In the case of a single family, we propose an IP-based exact procedure. Computational experiments based on randomly generated problem instances are carried out. It was demonstrated that the RKGA performs well under most of the experimental conditions. It was slightly outperformed by the GA-FFDn-SS approach in some situations. The proposed GAs reveal smaller E/T values than the simple heuristics. We also demonstrated that the IP-based algorithm works well for small- and medium-sized problem instances.

There are several directions for future search. First of all, we believe that it is possible to propose a RKGA-based heuristic that assigns orders to the early or on-time and the tardy order set and sequence them in each of the two sets. This can be carried out by considering random keys from the interval $[0, 2]$. Here, $\lfloor z_o \rfloor \in \{0, 1\}$ determines whether the order o belongs to the early or on-time set ($\lfloor z_o \rfloor = 0$) or to the tardy set ($\lfloor z_o \rfloor = 1$). The sequence of the orders in each of the two sets is determined by considering the fractional part $z_o - \lfloor z_o \rfloor$.

Moreover, it seems possible to study a similar scheduling problem for the item-processing environment as a second direction. As a third research direction, we are interested in considering MOJ scheduling problems with a nonrestrictive common due date and the E/T measure for parallel machines or even hybrid flow shops (*cf.* [39]).

As a fourth direction, we believe that it is worth to study MOJ scheduling problems for single and parallel machine environments where a restrictive common due date is considered.

Acknowledgements. We would like to thank Winfried Hochstättler for his hint to try 3-partition in the proof of Theorem 3.5.

References

[1] K.G. Agrawal and S.S. Heragu, A survey of automated material handling systems in 300-mm semiconductor fabs. *IEEE Trans. Semicond. Manuf.* **19** (2006) 112–120.

[2] C.J. Bean, Genetic algorithm and random keys for sequencing and optimization. *ORSA J. Comput.* **6** (1994) 154–160.

[3] C. Chien, S. Dauzère-Pérès, H. Ehm, J. Fowler, Z. Jiang, S. Krishnaswamy, L. Mönch and R. Uzsoy, Modeling and analysis of semiconductor manufacturing in a shrinking world: challenges and successes. *Eur. J. Ind. Eng.* **5** (2011) 254–271.

[4] V. Erramilli and J.S. Mason, Multiple orders per job compatible batch scheduling. *IEEE Trans. Electron. Packag. Manuf.* **29** (2006) 285–296.

[5] V. Erramilli and J.S. Mason, Multiple orders per job batch scheduling with incompatible jobs. *Ann. Oper. Res.* **159** (2008) 245–260.

[6] L. Foster and D. Pillai, Wafer logistics and automated material handling systems, in Handbook of Semiconductor Manufacturing Technology. Marcel Dekker Inc. (2000) 1067–1102.

[7] R.M. Garey and S.D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979).

[8] E.D. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading (1989).

[9] F.J. Gonçalves and C.G.M. Resende, Biased random-key genetic algorithms for combinatorial optimization. *J. Heuristics* **17** (2011) 487–525.

[10] V. Gordon, J.-M. Proth and C. Chu, A survey of the state-of-the-art of common due date assignment and scheduling research. *Eur. J. Oper. Res.* **139** (2002) 1–25.

[11] L.R. Graham, L.E. Lawler, K.J. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.* **5** (1979) 287–326.

[12] G.N. Hall and E.M. Posner, Earliness-tardiness scheduling problems I: weighted deviation of completion times about a common due date. *Oper. Res.* **39** (1991) 836–846.

[13] J. Jampani and J.S. Mason. Column generation heuristics for multiple machine, multiple orders per job scheduling problems. *Ann. Oper. Res.* **159** (2008) 261–273.

[14] J. Jampani, A.E. Pohl, J.S. Mason and L. Mönch, Integrated heuristics for scheduling multiple order jobs in a complex job shop. *Int. J. Metaheuristics* **1** (2010) 158–180.

[15] J. Jia and J.S. Mason, Semiconductor manufacturing scheduling of jobs containing multiple orders on identical parallel machines. *Int. J. Prod. Res.* **47** (2009) 2565–2585.

[16] J.J. Kanet, Minimizing the average deviation of job completion times about a common due date. *Nav. Res. Logist.* **28** (1981) 643–651.

[17] D.J. Laub, W.J. Fowler and B.A. Keha, Minimzing makespan with multiple-orders-per-job in a two-machine flowshop. *Eur. J. Oper. Res.* **128** (2007) 63–79.

[18] X. Li, H. Chen, R. Xu and X. Li, Earliness–tardiness minimization on scheduling a batch processing machine with non-identical job sizes. *Comput. Ind. Eng.* **87** (2015) 590–599.

[19] S. Martello and P. Toth, Knapsack Problems: Algorithms and Computer Implementations. Wiley & Sons, Chichester (1990).

[20] J.S. Mason and J.-S. Chen, Scheduling multiple orders per job in a single machine to minimize total completion time. *Eur. J. Oper. Res.* **207** (2010) 70–77.

[21] J.S. Mason, P. Qu, E. Kutanoglu and W.J. Fowler, *The Single Machine Multiple Orders per Job Scheduling Problem.* Technical Report, ASUIE-ORPS-2004-04, Arizona State University, Tempe (2004).

[22] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, 3rd edition. Springer, Berlin (1996).

[23] L. Mönch and R. Unbehaun, Decomposition heuristics for minimizing earliness-tardiness on parallel burn-in ovens with a common due date. *Comput. Oper. Res.* **34** (2007) 3380–3396.

[24] L. Mönch, R. Unbehaun and I.Y. Choung, Minimizing earliness and tardiness on a single burn-in oven with a common due date and a maximum available tardiness constraint. *OR Spectr.* **28** (2006) 177–198.

[25] L. Mönch, W.J. Fowler, S. Dauzère-Pérès, J.S. Mason and O. Rose, A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *J. Sched.* **14** (2011) 583–595.

[26] L. Mönch, J. Zimmermann, J.S. Mason and W.J. Fowler, Multiple orders per job formation and release strategies in large scale wafer fabs: a simulation study. *J. Simul.* **5** (2011) 25–43.

[27] L. Mönch, W.J. Fowler and J.S. Mason, Production Planning and Control for Wafer Fabrication Facilities: Modeling, Analysis, and Systems. Springer, New York (2013).

[28] L. Mönch, R. Uzsoy and W.J. Fowler. A survey of semiconductor supply chain models part I: semiconductor supply chains, strategic network design, and supply chain simulation. To appear in: *Int. J. Prod. Res.* Doi: 10.1080/00207543.2017.1401233 (2017).

[29] J.R. Montoya-Torres, A literature survey on the design approaches and operational issues of automated wafer-transport systems for wafer fabs. *Prod. Plan. Control* **17** (2006) 648–663.

[30] R.N. Parsa, B. Karimi and S.M. Moattar Husseini, Exact and heuristic algorithms for the just-in-time scheduling problem in a batch processing system. *Comput. Oper. Res.* **80** (2017) 173–183.

[31] M. Pinedo, Scheduling: Theory, Algorithms, and Systems, 5th edition. Springer, New York (2016).

[32] P. Qu and S. Mason, Metaheuristic scheduling of 300-mm lots containing multiple orders. *IEEE Trans. Semicond. Manuf.* **18** (2005) 633–643.

[33] A. Rogers and A. Prügel-Bennett, Modelling the dynamics of a steady state genetic, in Vol. 5 of Foundations of Genetic Algorithms, edited by W. Banzhaf and C. Reeves. Springer (1999) 57–68.

[34] C.S. Sarin, L. Wang and M. Cheng, A single-machine, single-wafer-processing, multiple-lots-per-carrier scheduling problem to minimize the sum of lot completion times. *Comput. Oper. Res.* **39** (2012) 1411–1418.

[35] C.S. Sarin, L. Wang and M. Cheng , Minimising makespan for a two-machine, flow shop, single-wafer-processing, multiple-jobs-per-carrier scheduling problem. *Int. J. Plan. Sched.* **1** (2012) 171–208.

[36] O. Sobeyko and L. Mönch, Genetic algorithms to solve a single machine multiple orders per job scheduling problem, in *Proc. of the 2010 Winter Simulation Conference* (2010) 2493–2503.

[37] O. Sobeyko and L. Mönch, Grouping genetic algorithms for solving single machine multiple orders per job scheduling problems. *Ann. Oper. Res.* **235** (2015) 709–739.

[38] G. Syswerda, A study of reproduction in generational and steady-state genetic algorithms, in Vol. **1** of Foundations of Genetic Algorithms, edited by E.J.G. Rawlins. Morgan Kaufmann Publishers (1991) 94–101.

[39] Y. Tan, L. Mönch and W.J. Fowler, A hybrid scheduling approach for a two-stage flexible flow shop with batch processing machines. *J. Sched.* **21** (2018) 209–226.

[40] F. Tanrisever and E. Kutanoglu, Forming and scheduling jobs with capacitated containers in semiconductor manufacturing: single machine problem. *Ann. Oper. Res.* **159** (2008) 5–24.

[41] M. Wall, Galib: A C++ Library of Genetic Algorithms Components. Available at Website: http://lancet.mit.edu/ga/ (2017).

[42] F. Wilcoxon, Individual comparisons by ranking methods. *Biometrics* **1** (1945) 80–83.