



ELSEVIER

Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

Computers & Operations Research 32 (2005) 2731–2750

computers &
operations
research

www.elsevier.com/locate/dsw

Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times

Lars Mönch^{a,*}, Hari Balasubramanian^b, John W. Fowler^b, Michele E. Pfund^b

^a*Institute of Information Systems, Technical University of Ilmenau, 98684 Ilmenau, Germany*

^b*Department of Industrial Engineering, Arizona State University, Tempe, AZ 85287, USA*

Available online 28 May 2004

Abstract

This research is motivated by a scheduling problem found in the diffusion and oxidation areas of semiconductor wafer fabrication, where the machines can be modeled as parallel batch processors. We attempt to minimize total weighted tardiness on parallel batch machines with incompatible job families and unequal ready times of the jobs. Given that the problem is NP-hard, we propose two different decomposition approaches. The first approach forms fixed batches, then assigns these batches to the machines using a genetic algorithm (GA), and finally sequences the batches on individual machines. The second approach first assigns jobs to machines using a GA, then forms batches on each machine for the jobs assigned to it, and finally sequences these batches. Dispatching and scheduling rules are used for the batching phase and the sequencing phase of the two approaches. In addition, as part of the second decomposition approach, we develop variations of a time window heuristic based on a decision theory approach for forming and sequencing the batches on a single machine.

© 2004 Elsevier Ltd. All rights reserved.

Keywords: Scheduling; Batching; Parallel machines; Genetic algorithms; Semiconductor manufacturing

1. Introduction

Wafer fabrication, the first step in semiconductor manufacturing, is characterized by hundreds of steps, re-entrant flows, sequence-dependent setups, diversity of product mix and batch processing. This complexity involved, combined with meeting customer due dates with different priorities—an important criterion for customer satisfaction—is a difficult task. This research focuses on the

* Corresponding author.

E-mail addresses: lars.moench@tu-ilmenau.de (L. Mönch), hari.balasubramanian@asu.edu (H. Balasubramanian), john.fowler@asu.edu (J.W. Fowler), michele.pfund@asu.edu (M.E. Pfund).

scheduling of batch processing machines found in the diffusion and oxidation areas of a wafer fabrication facility. The processing times of these batching operations can be extremely long (10 h) when compared to other operations (1–2 h) in a wafer fab. Mehta and Uzsoy [1] state that the effective scheduling of these operations is important to achieving good system performance. Though several jobs can be processed simultaneously in these batch processing machines, process restrictions require that only jobs belonging to the same family be processed together at one time. In addition, the jobs to be processed have different priorities/weights, due dates and ready times. In the presence of unequal ready times, it is sometimes advantageous to form a non-full batch while in other situations it is a better strategy to wait for future job arrivals in order to increase the fullness of the batch.

We model diffusion and oxidation operations as parallel batch processing machines with incompatible job families. The performance measure to be minimized is total weighted tardiness. Total weighted tardiness ($w_j T_j$) is the summation of the weighted tardiness over all jobs $j = 1, 2, 3, \dots, n$, where w_j is the weight (priority) of job j , C_j is the completion time and d_j the due date of job j and $T_j = \max(0, C_j - d_j)$. Furthermore, we consider the more realistic case of dynamic job arrivals, i.e., we allow the ready times of the jobs to reflect differences in arrival times of the jobs. The batch problem for parallel machines is more complex than the single machine problem $1||\sum w_j T_j$, which is NP-hard by Lawler [2]. Therefore, this research proposes two heuristic approaches that lead to good solutions.

The paper is organized as follows. We discuss related work in Section 2. The statement of the problem is provided in Section 3. We describe the suggested heuristics in Section 4. In Section 5, we describe the implementation of the genetic algorithm-based heuristics and Section 6 outlines the algorithms considered. We continue with parameter setting and test data generation in Sections 7 and 8, respectively. In Section 9, we describe the computational experiments and their results, and finally provide conclusions and future research in Section 10.

2. Previous related work

Batching problems in manufacturing have been revisited by many researchers. Perez et al. [3] provide a detailed review and classification of papers that have dealt with deterministic scheduling problems related to batching. Azizoglu and Webster [4] consider scheduling a single batch-processing machine with incompatible job families where the jobs have arbitrary sizes and weights to minimize total weighted completion time ($1|batch, non-identical, incompatible|\sum w_j C_j$) (we will use the $\alpha|\beta|\gamma$ notation of Graham et al. [5] throughout the rest of the paper). They use a branch and bound procedure. Dupont and Dhaenens-Flipo [6] develop a branch and bound procedure using dominance rules for the problem of scheduling a single batch-processing machine with compatible families in which the jobs have non-identical capacity requirements to minimize the makespan ($1|batch, non-identical|C_{max}$). Wang and Uzsoy [7] consider the problem of scheduling a single batch machine with job release dates and compatible families to minimize maximum lateness ($1|batch, r_j|L_{max}$). They use a genetic algorithm coupled with dynamic programming techniques to solve the problem.

Mehta and Uzsoy [1], Perez et al. [3], and Devpura et al. [8] deal with the single machine batching problem with incompatible job families. Mehta and Uzsoy [1] consider the objective of minimizing

total tardiness ($1|batch, incompatible|\Sigma T_j$), and suggest a decomposition approach. The problem is decomposed into two parts: batching the jobs, and sequencing the batches. Perez et al. [3] extend this work to minimizing total weighted tardiness ($1|batch, incompatible|\Sigma w_j T_j$). Devpura et al. [8] suggest a pairwise interchange heuristic that changes the composition of batches to improve on the total weighted tardiness value of the schedule. In Balasubramanian et al. [9] and Mönch et al. [10], the authors extend the existing solution methods for the problem $1|batch, incompatible|\Sigma w_j T_j$ to the $P_m|batch, incompatible|\Sigma w_j T_j$ problem.

However, in real-world situations, it is useful to consider future job arrivals. Glassey and Weng [11], Fowler et al. [12,13], Robinson et al. [14], Cigolini et al. [15], and Duenyas and Neale [16] illustrate the need to use information on future arrivals for making decisions in a stochastic batch-processing environment. Knowledge of jobs arriving in the future can be obtained from the manufacturing execution system on the shopfloor. The use of this information can lead to decisions better than those taken by considering only the current state of the system. Based on this intuitive idea, new heuristics can be devised where batches are formed and sequenced for a pre-specified time window using a close-to-optimal procedure. Chand et al. [17] considered a time window approach for the $1|r_j|\Sigma C_j$ problem while Mason et al. [18] suggest a modification of the Apparent Tardiness Cost index to cover batching problems with ready times.

In this paper, we extend the solution method for parallel machine batching problems considered by Balasubramanian et al. [9] and Mönch et al. [10] to the considerably more difficult, but more realistic, case of jobs with unequal ready times. Unequal ready times make both the batch formation and subsequent scheduling stage more complex.

3. Problem assumptions and notation

Assumptions involved in the scheduling of parallel batch processing machines with incompatible job families and unequal ready times of the jobs to minimize total weighted tardiness include:

1. Jobs of the same family have the same processing times.
2. All the batch processing machines are identical in nature.
3. Once a machine is started, it cannot be interrupted, i.e. no pre-emption is allowed.

We use the following notation throughout the rest of the paper.

1. Jobs fall into different incompatible families that cannot be processed together. There exist f such families.
2. There are n jobs that have to be scheduled.
3. There are m identical machines in parallel.
4. There are n_j jobs of family j to be scheduled: $\sum_{j=1}^f n_j = n$.
5. Job i of family j is represented as ij .
6. The priority weight for job i of family j is represented as w_{ij} .
7. The due date of job i of family j is represented as d_{ij} .
8. The processing time of jobs in family j is represented as p_j .
9. The ready time of job i in family j is represented as r_{ij} .

10. The batch processing machine capacity is B jobs.
11. Batch k of family j is represented by B_{kj} .
12. The completion time of job i of family j is denoted by C_{ij} .
13. The weighted tardiness of job ij is represented as $w_{ij}T_{ij} = w_{ij}(C_{ij} - d_{ij})^+$, where we use the notation $x^+ := \max(x, 0)$ throughout the rest of the paper.

Using the $\alpha|\beta|\gamma$ notation, this problem can be represented as:

$$P_m|r_j, batch, incompatible|\Sigma w_j T_j.$$

4. Decomposition approaches for parallel machine scheduling

4.1. Description

Due to the complexity of the problem, a three-stage approach and a two-stage approach are taken. The first decomposition approach (Approach 1) forms the batches in the first stage, assigns the batches to the machines in the second stage, and sequences the batches on each of the machines in the third stage. Approach 1 is similar to a decomposition scheme suggested by Mehta and Uzsoy [1]. The second decomposition approach (Approach 2) assigns the jobs to the machines in the first stage, forms batches on each machine, and sequences these batches simultaneously on their respective machines in the second stage.

4.2. Using genetic algorithms for parallel machine scheduling

Genetic algorithms have been used in many combinatorial optimization problems, especially assignment and scheduling problems. We refer to the recent survey paper of Dimipoulos and Zalzalá [19] for using genetic algorithms in manufacturing. Aytug et al. [20] also provide an extensive survey and classification of genetic algorithms applied to production and operations management problems. Until recently, GA methods seemed to be too costly in comparison to dispatching methods. However, with the recent dramatic increase in computer efficiency GA, methods have become more competitive. Gravel et al. [21] use a double-loop genetic algorithm for a parallel machine scheduling problem within an aluminum foundry. Fowler et al. [22] applied genetic algorithms to solve parallel machine scheduling problems with sequence-dependent setup times. They proposed a hybridized genetic algorithm. The GA is used to assign jobs to the parallel machines, and after the assignment phase, the single machine sub-problems are solved by using dispatching rules. Mönch et al. [10] applied heuristics using genetic algorithms to the $P_m|batch, incompatible|\Sigma w_j T_j$ problem.

Similarly, in this research, a GA is used for the assignment of either jobs or batches to the machines. In approach 1, the GA assigns batches to machines, while in approach 2, the GA assigns jobs to machines. After the assignment phase, the GA evaluates each of these assignments by aggregating the total weighted tardiness values of the single machine sequences (obtained on each machine by dispatching and scheduling rules) on all the machines. The GA converges towards assignments that give good solutions with respect to total weighted tardiness (Figs. 1 and 2).

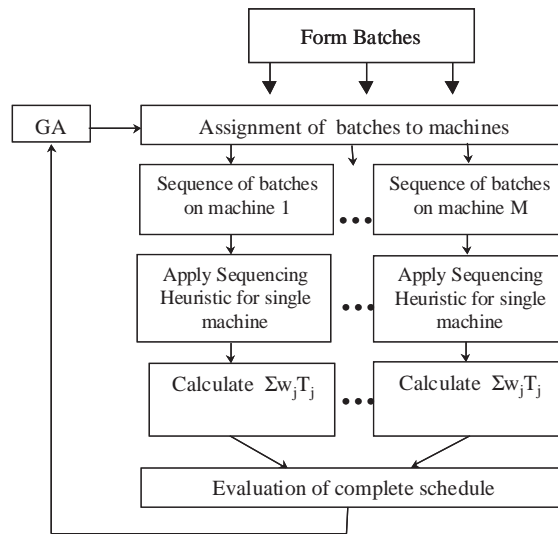


Fig. 1. Approach 1.

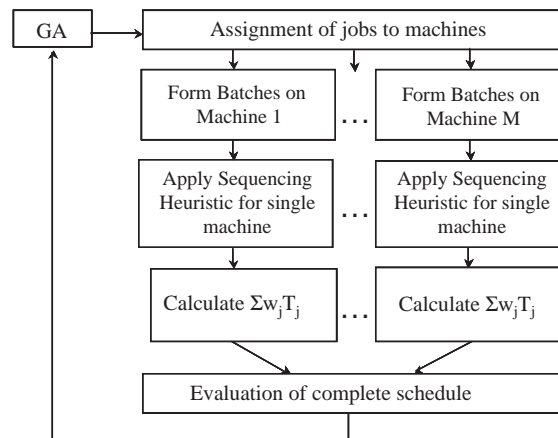


Fig. 2. Approach 2.

The GA is used for assignment purposes only; it is hybridized with modifications of different heuristics for batching and sequencing as suggested in Fowler et al. [22], and used in Balasubramanian et al. [9] and Mönch et al. [10] for the $P_m | batch, incompatible | \sum w_j T_j$ problem. Thus, two different versions of the GA are developed: GA Version 1 (based on approach 1) and GA Version 2 (based on approach 2).

4.3. Batching and sequencing heuristics

4.3.1. Modification of the ATC dispatching rule

The well-known apparent tardiness cost (ATC) heuristic suggested by Vepsalainen and Morton [23] is used in several ways in this research.

1. *ATC used independently as a dispatching rule to solve the parallel machine scheduling problem.*

This approach does not consider assignment of jobs or batches to the machines at the outset, and therefore does not require the GA. Instead, at every point of time t when a machine becomes free, one batch from each family is chosen and of all the considered batches one is chosen and scheduled on the machine. We consider a time window $(t, t + \Delta t)$. We denote by

$$M(j, t, \Delta t) := \{ij | r_{ij} \leq t + \Delta t\}$$

the set of unscheduled jobs of family j with arrival (ready) time less than the upper boundary of the time window interval. Then we derive the set

$$\tilde{M}(j, t, \Delta t, thres) := \{ij | ij \in M(j, t, \Delta t) \text{ and } pos(ij) < thres\}.$$

Here, we denote by I_{ij} a criterion for evaluating the jobs and by $pos(ij)$ the position of job ij with respect to I_{ij} . By $thres$ we denote the maximum number of jobs of $\tilde{M}(j, t, \Delta t, thres)$ to be considered. We consider all batch combinations on the job set $\tilde{M}(j, t, \Delta t, thres)$. We define $c := card(\tilde{M}(j, t, \Delta t, thres))$. If $c > B$ then we have to consider

$$\binom{c}{B} + \binom{c}{B-1} + \dots + \binom{c}{1}$$

different batches for the next batch to be formed. Obviously, the computational effort for evaluating all batch combinations depends strongly on the choice of the parameters Δt and $thres$. Our time window approach that reduces the number of jobs for forming batch combinations is similar to an approach of Ovacik and Uzsoy [24] used in a sequence-dependent setup minimization situation.

We continue with describing a criterion for the calculation of $pos(ij)$. The criterion is based on the ATC index. The ATC index $I_{ij,ATC}$ for job i belonging to family j calculated at time t is given below:

$$I_{ij,ATC}(t) = \left(\frac{w_{ij}}{p_j} \right) \exp \left(- \frac{(d_{ij} - p_j + (r_{ij} - t))^+}{k \bar{p}} \right).$$

In $I_{ij,ATC}$, k is a look-ahead parameter and \bar{p} is the average processing time of the remaining unscheduled jobs. We sort the jobs of the set $M(j, t, \Delta t)$ in a non-decreasing order according to $I_{ij,ATC}$ and take the first $thres$ of them in order to form the set $\tilde{M}(j, t, \Delta t, thres)$.

For the evaluation of a certain batch combination we consider the following three indices:

$$I_{bj1}(t) = \left(\frac{w_{bj}}{p_j} \right) \exp \left(- \frac{(d_{bj} - p_j + (r_{bj} - t))^+}{k \bar{p}} \right) \min \left(\frac{n_{bj}}{B}, 1 \right),$$

where $d_{bj} = \min_{i \in B_{bj}}(d_{ij})$: minimum due date among the jobs contained in the batch, $r_{bj} = \max_{i \in B_{bj}}(r_{ij})$: maximum ready time of the jobs in the batch, w_{bj} the average weight of the jobs that form the batch, and n_{bj} the number of jobs in the batch.

This rule was suggested in a slightly different form by Mason et al. [18].

The second index used in this study is given by

$$I_{bj2}(t) = \sum_{i=1}^{n_{bj}} \left(\frac{w_{ij}}{p_j} \right) \exp \left(- \frac{(d_{ij} - p_j - t + (r_{bj} - t)^+)^+}{k \bar{p}} \right) \min \left(\frac{n_{bj}}{B}, 1 \right).$$

This rule is motivated by the fact, that if the jobs for a batch to be formed are not available then the priority should be reduced. In the rule, we therefore add the quantity $(r_{bj} - t)^+$ to the slack. On the other hand, if all jobs are available at time t then there is no reason for taking the ready time into account. Because we take the sum over all jobs in the batch, the rule tries to increase the fullness of the batches.

The third rule included in our heuristics is an extension of the X-RM rule suggested by Morton and Ramnath [25] to the batch case. The priority index of this rule is given by

$$I_{bj3}(t) = \left(\frac{w_{bj}}{p_j} \right) \exp \left(- \frac{(d_{bj} - p_j - t)^+}{k \bar{p}} \right) \left(1 - \frac{X}{\tilde{p}} (r_{bj} - t)^+ \right) \min \left(\frac{n_{bj}}{B}, 1 \right),$$

where X is a scaling factor and \tilde{p} is the average processing time of all jobs to be processed. The third term reduces the priority if at least one job of the batch is not available at time t , where the fourth term reduces the priority due to non-fullness of the batch. The X scaling factor works similar to the k value in I_{bj1} and I_{bj2} . It scales the difference between the ready time and the time for decision making. We denote the three different rules by batched apparent tardiness cost (BATC)-I, BATC-II, and BATC - III, respectively.

The steps of the different BATC-based dispatching heuristics can be summarized as below:

1. At time t , choose a window size Δt and choose at most *thres* jobs from the set $M(j, t, \Delta t)$. Calculate the corresponding I_{ij} indexes for each family with unscheduled jobs and sort them in decreasing order. The result is the set $\tilde{M}(j, t, \Delta, \text{thres})$.
2. Select a machine m with ready time either less than or equal to t .
3. Compute the BATC index for all batch combinations in each family, and select the batch b with highest BATC index.
4. Schedule the selected batch b on machine m .
5. Advance the ready time of the machine m to $\max(t, r_{bj}) + p_b$, where p_b is the processing time of the family to which batch b belongs.
6. If all jobs are scheduled go to 7, else go to 2.
7. Stop.

2. Using ATC for forming batches

Note that when the seven steps of the BATC heuristics just described are carried out, we have a complete schedule of jobs in batches and batches sequenced on machines. Clearly, this heuristic could be used for batching if we consider the machines unscheduled, and use only the batches that are formed. These batches have been formed dynamically (not at time $t = 0$), and therefore could

be an effective way of batching the jobs. This technique of batching is used in approach 1, which requires batches to be formed in the first stage. The main steps are described below:

- (a) run one of the BATC heuristics independently and form batches,
- (b) assign the batches formed in step 1 to a machine using the GA,
- (c) sequence the batches on their respective machines using BATC.

In the second approach, the BATC rule is used to batch the jobs only after they have been assigned to the machines. The basic steps of this approach are described below:

- (a) assign jobs to machines using the GA,
- (b) on each machine, form and sequence batches using one of the BATC-based rules.

3. Using the BATC dispatching rule as a sequencing rule

BATC can be used as a sequencing rule for a single machine. At any time t , for all the batches one of the indexes $I_{bjl}(t)$, $l = 1, 2, 3$, is computed. The batch with the maximum index is scheduled, time t is advanced by the processing time and the maximum of the ready time of the batch and t , and all the BATC indices are recalculated. This is continued till all the batches are scheduled.

As the BATC-type rules were found to be sensitive to the k -value setting, we used ten different k -values from 0.5 and 5 in increments of 0.5 (similar to Mehta and Uzsoy [1]). For a particular test instance, we run the BATC independently for each k -value, calculate the total weighted tardiness value, and choose the k -value that gives the minimum total weighted tardiness. This k -value was then fixed (for a given test instance) for algorithms of both approach 1 and approach 2 that use BATC-based rules. The scaling parameter X of the BATC-III dispatching rule is fixed at 1.6 as suggested by Akturk and Ozdemir in [26].

4.3.2. Decision theory approach for forming and sequencing batches

In this section, we describe a procedure that can be used for solving the $1|r_j$, batch, incompatible $|\Sigma w_j T_j$ problem. Of course, the suggested algorithm can be used in second and third phase of approach 2 for the corresponding parallel machine scheduling problem. This is an extension of the decision theory approach of Kanet and Zhou [27] to the case of batching with incompatible families.

For the description of the algorithm we first fix a time t . We define sets $\tilde{M}(j, t, \Delta t, thres)$ for each family j . Then we form batches in each of the sets. Furthermore, we denote by $M := \cup_{k=1}^f \tilde{M}(k, t, \Delta t, thres)$ the set of jobs available in $(t, t + \Delta t)$. We calculate the expected tardiness $TWT(B_j)$ with respect to the chosen batch first. We denote this batch by B_j . We express the tardiness by the following expression:

$$TWT(B_j) := \sum_{ij \in B_j} w_{ij}(t^* + p_j - d_{ij})^+ + \sum_{ki \in M - B_j} w_{ki}(t^* + p_j + p_i + p^* - d_{ki})^+,$$

where $t^* := \max(t, \max_{ij \in B_j}(r_{ij}))$ is the time where batch B_j starts processing. We define

$$p^* := \frac{1}{2B} \sum_{ki \in M - B_j} p_i.$$

In the $TWT(B_j)$ expression, the first term represents the weighted tardiness of the chosen batch. The second term represents a rough estimate for the total weighted tardiness of the rest of the jobs. In the above expression for $TWT(B_j)$ job ki starts processing only after batch B_j and a delay of p^* . The delay p^* implies that job ki may not start processing immediately following B_j as other jobs may be scheduled before it. The term p^* depends on the total number of unscheduled jobs in the time window. If the number of unscheduled jobs is high then p^* will have a large value; if there are few unscheduled jobs, p^* will have a small value.

We calculate the quantity $TWT(B_j)$ for all batches B_j and choose the batch with the smallest value for $TWT(B_j)$. Of course, the quality of decision making using the window technique depends on the quality of estimates for the completion times of the jobs.

The advantage of the decision theory approach is its ability to evaluate the impact of a decision in its immediate time “vicinity”. Not only the attributes of the jobs (such as weights, release dates, due dates, slack, processing time) are considered, but also the effect of the decision on the unscheduled jobs already present and those arriving in the immediate future is projected. This ability of the decision theory approach sets it apart from the ATC-based rules that consider only the attributes of the jobs and the selected batch to make a decision.

5. Implementation of the GA

In this research, the GA is used to assign jobs or batches to machines. The procedure is similar to that used by Fowler et al. [22], Gravel et al. [21], and Mönch et al. [10]. In all these papers, the assignment of jobs to machines is done using a GA. The following steps describe how the GA is used in this research: coding, initialization, selection, crossover, mutation, evaluation and termination scheme. Since the GA can be used to assign batches or jobs, the terms entity or entities are used to represent both.

1. *Coding*: We use an entity-based representation. In the case of k different entities, i.e., batches or jobs, the following representation is used for assignment:

$$c := (m_1 \ m_2 \ \cdots \ m_{k-1} \ m_k),$$

where m_j represents the machine that is used for processing entity j . For example $c := (1 \ 10 \ 4 \ 4 \ 4)$ means that entity 1 will be processed on machine 1, entity 2 on machine 10, and the remaining entities on machine 4. We call each of these representations a single chromosome. Our genetic algorithm maintains a population of these chromosomes.

2. *Initialization*: Each chromosome, represented by an array as described in the coding part, is randomly initialized by assigning a random number from the set $\{1, \dots, m\}$ to each of the k entities.
3. *Parents selection*: For selection, the roulette wheel technique is followed (see Michalewicz [28], Wall [29]). The probability of selection of a certain chromosome is proportional to its fitness; the fitness is calculated using linear scaling. The fitness function of the GA is provided by the total weighted tardiness measure of the schedule.
4. *Crossover*: A one-point crossover [28] is utilized in which two parent chromosomes are selected randomly according to their fitness and the crossover point is randomly chosen. We

denote the first chromosome by $c_1 := (m_{11} \ m_{12} \ \dots \ m_{1k-1} \ m_{1k})$ and the second chromosome by $c_2 := (m_{21} \ m_{22} \ \dots \ m_{2k-1} \ m_{2k})$. After performing one point crossover (with crossover point at position s) we obtain the two resulting chromosomes given by $c_3 := (m_{21} \ \dots \ m_{1s} \ \dots \ m_{1k})$ and $c_4 = (m_{11} \ \dots \ m_{2s} \ \dots \ m_{2k})$.

5. *Mutation*: For each gene of each of the resulting offspring, a gene of the chromosome is randomly changed to a different machine number (flip operator) according to a pre-defined probability.
6. *Choosing a new generation*: A steady-state genetic algorithm with overlapping populations is used. The worst elements of the preceding population are replaced. The number of replaced elements depends on the given replacement probability.
7. *Stopping criterion*: The genetic algorithm is controlled by a prescribed number of generations and by using a diversity measure to stop the algorithm. The diversity of the algorithm is defined by the standard deviation of the fitness values of all chromosomes of a population in a particular generation. During the execution of the GA, if the diversity of the population at a certain generation is smaller than a given threshold value or if the number of generations is higher than the prescribed limit, the GA terminates.
8. The GA is implemented based on the object-oriented C++ framework Galib [29] that supports the chromosome representations, the used genetic operators, and the scheme of the GA.

6. Algorithms

A number of algorithms are available with which both the first and the second approach can be hybridized. For batching we can use any of the BATC-I, BATC-II, and BATC-III heuristics. For the single machine batch-sequencing subproblems, we can use any of the BATC-I, BATC-II, and BATC-III heuristics. Based on these, we propose three algorithms for approach 1, denoted by BATC-I_GA-1_BATC-I, BATC-II_GA-1_BATC-II, and BATC-III_GA-1_BATC-III.

For the second approach, we can use any of the three BATC dispatching rules for forming and sequencing the batches. We denote the corresponding algorithms by GA-2_BATC-I, GA-2_BATC-II, and GA-2_BATC-III.

In addition, we also incorporate the decision-theory heuristic into the second approach. We call this algorithm GA-2_DTH. In Table 1, we summarize the different algorithms.

Table 1
Algorithms

Algo. no.	Phase 1	Phase 2	Phase 3
1	BATC-I	GA 1	BATC-I
2	BATC-II	GA 1	BATC-II
3	BATC-III	GA 1	BATC-III
4	GA 2	BATC-I	—
5	GA 2	BATC-II	—
6	GA 2	BATC-III	—
7	GA 2	DTH	—

Table 2
Parameter settings GA approach 1

Parameter	Value
Population size	300
Crossover probability	0.80
Mutation probability	0.03
Replacement probability	0.60
Diversity	0.01
Number of generations	500

Table 3
Parameter settings approach 2

Parameter	Value
Population size	200
Crossover probability	0.80
Mutation probability	0.01
Replacement probability	0.60
Diversity	0.03
Number of generations	1000

7. Parameter settings for the GA

We choose the parameters of the genetic algorithm by trial and error based on extensive computational experiments. From preliminary experiments it was determined that the population size and the mutation probability have the largest impact on the solution quality. Therefore, we apply a grid search on the mutation probability values $\{0.1, 0.2, 0.3\}$ and on the population size $\{200, 250, 300\}$ for the two approaches. It was also observed that the computation time depended primarily on the number of generations. A large number of generations are especially required for approach 2 where the solution space is large and thus the convergence of the algorithm is slow. Therefore, we choose for the first approach a generation number of 500 and for the second approach a generation number of 1000. The remaining parameters are fixed as described in Tables 2 and 3. We apply a specific GA parameter combination to each of the 162 factor combinations documented in Table 4. Then we choose the GA parameter combination that leads to the best average total weighted tardiness result.

8. Test data generation

For testing the heuristic approaches random instances were generated in a manner similar to Akturk and Ozdemir [26]. We consider the case of 3, 4, and 5 parallel machines. The weights w_j are chosen from a uniform distribution over $(0, 1)$. We generate ready times r_j from a uniform distribution between 0 and $\alpha/(mB) \sum_{j=1}^n p_j$. In the next step, we generate slack times between due dates and earliest completion time from a uniform distribution from 0 to $\beta/(mB) \sum_{j=1}^n p_j$. The earliest completion time of job j is given by $r_j + p_j$. Here, α and β are specific numbers from $(0, 1]$

Table 4
Test date parameters

Problem parameter	Values used	Total values
Number of machines	3,4,5	3
Number of jobs/family	60,80,100	3
Batch size	4,8	2
Number of families	3	1
Family processing time	2 with a probability of 0.2 4 with a probability of 0.2 10 with a probability of 0.3 16 with a probability of 0.2 20 with a probability of 0.1	1
Weight per job w_{ij}	Uniform (0,1)	1
Release date r_{ij}	$r_{ij} \sim \text{Uniform}(0, \alpha/mB\sum p_j)$	9
Due date d_{ij}	$d_{ij} - r_{ij} \sim \text{Uniform}(0, \beta/mB\sum p_j)$ $\alpha = 0.25, 0.50, 0.75$ $\beta = 0.25, 0.50, 0.75$	
	Total parameter combinations	162
	Number of problem instances	10
	Total problems	1620

chosen in our experiments to control the distribution of ready times and the due dates. The value of α controls the range of the ready times, while the value of β controls the due date range. Higher values of α lead to more spread out release dates. On the other hand, higher values of β result in more loose due dates.

We consider the case of three incompatible job families. The processing times for each family are chosen as described in Table 4. Note that we try to mimic the fact that processing times on batch machines are usually long compared to processing times on other machines of a wafer fab. We consider 10 stochastically independent replications for each fixed parameter combination; therefore, in all, we consider 1620 different test cases. We use three independent replications of a GA run for each test instance. The resulting performance measure values are the average values for the individual results for each replication of the genetic algorithm.

9. Results

The average results are summarized in Table 5. They have been presented as follows:

1. Instead of comparing all cases individually, the cases were grouped according to levels of factors. For example, the results for $\alpha = 0.50$ are for all runs with $\alpha = 0.50$ while all other factors have been varied at their different levels.
2. For a particular factor in the table the value provided is the average of the ratios of total weighted tardiness value of an algorithm with the total weighted tardiness value of the BATC-II rule (used

Table 5
Results for the different algorithms ($\Delta t = 4$; $thres = 10$)

Compare	BATC-II	BATC-I GA 1 BATC-I	BATC-II GA 1 BATC-II	BATC-III GA 1 BATC-III	GA 2 BATC-I —	GA 2 BATC-II —	GA 2 BATC-III —	GA 2 DTH —
Machines								
$m = 3$	1.000	2.6560	0.9087	1.5016	1.8226	0.9221	1.2541	0.8192
$m = 4$	1.000	2.0312	0.9158	1.3980	1.5263	0.9620	1.1449	0.8742
$m = 5$	1.000	1.8336	0.9301	1.4074	1.5174	1.0396	1.1977	1.0034
Batch size								
$B = 4$	1.000	2.2383	0.9004	1.5303	1.9193	0.9725	1.3479	0.8599
$B = 8$	1.000	2.1089	0.9360	1.3410	1.3249	0.9767	1.0499	0.9379
Release time factor								
$\alpha = 0.25$	1.000	1.3995	0.9395	1.2127	1.3399	0.9576	1.1125	0.9342
$\alpha = 0.50$	1.000	2.7057	0.8947	1.5648	2.0523	0.9863	1.3640	0.8421
$\alpha = 0.75$	1.000	2.4157	0.9203	1.5296	1.4741	0.9798	1.1203	0.9204
Due date factor								
$\beta = 0.25$	1.000	1.2238	0.9338	1.0899	1.0637	0.8917	0.8998	0.8902
$\beta = 0.50$	1.000	2.2241	0.9103	1.4678	1.5046	0.9543	1.1248	0.8859
$\beta = 0.75$	1.000	3.0730	0.9105	1.7493	2.2981	1.0777	1.5721	0.9207
Overall	1.000	2.1736	0.9182	1.4357	1.6221	0.9746	1.1989	0.8989

independently without the GA). This was chosen because it performed best overall between BATC-I, BATC-II, and BATC-III when used as a dispatching rule. In order to assess the performance of BATC with a common used dispatch policy, we compare BATC II against an earliest due date (EDD) dispatching policy. It turns out that using BATC II leads to schedules that have a total weighted tardiness that is on average 20% of the total weighted tardiness of the corresponding schedule obtained by EDD.

Table 5 indicates that three of the seven algorithms outperform the BATC-II rule, namely the algorithms BATC-II_GA-1_BATC-II, GA-2_BATC-II, and GA-2_DTH. The algorithm GA-2_DTH leads to a TWT improvement of about 10%, while the algorithm BATC-II_GA-1_BATC-II leads to an average improvement of about 9%. In Table 6, we present the worst-case results for the different algorithms. A significant fact to be noted is that even in the worst case, BATC-II_GA-1_BATC-II and GA-2_DTH are on average more than 7% better than BATC-II used independently. From Tables 5 and 6 we conclude the strong dependency of TWT improvement due to parameters α and β . Therefore, we present a more detailed analysis of this impact in Section 9.1.

9.1. Effect of ready time range (α) and due date range (β) on solution quality

As described above, the parameters α and β have strong an impact on the TWT improvement of the algorithms. Therefore, we consider all the possible α and β parameter combinations in Table 7.

Table 6
Worst case results for the different algorithms ($\Delta t = 4$; $thres = 10$)

Compare	BATC-II	BATC-I GA 1 BATC-I	BATC-II GA 1 BATC-II	BATC-III GA 1 BATC-III	GA 2 BATC-I —	GA 2 BATC-II —	GA 2 BATC-III —	GA 2 DTH —
Machines								
$m = 3$	1.000	3.0527	0.9210	1.6113	2.0579	1.0002	1.5046	0.8311
$m = 4$	1.000	2.5629	0.9301	1.5198	2.0990	1.1052	1.4087	0.8720
$m = 5$	1.000	1.9282	0.9345	1.5149	1.7852	1.1445	1.3319	1.0543
Batch size								
$B = 4$	1.000	2.8177	0.9089	1.7806	2.4637	1.0928	1.6979	0.8483
$B = 8$	1.000	2.2115	0.9482	1.3167	1.4976	1.0739	1.1322	0.9899
Release time factor								
$\alpha = 0.25$	1.000	1.3635	0.9365	1.1840	1.3289	0.9432	1.0946	0.9149
$\alpha = 0.50$	1.000	3.5656	0.9137	1.8128	2.8229	1.1480	1.8362	0.8976
$\alpha = 0.75$	1.000	2.6148	0.9354	1.6492	1.7902	1.1587	1.3144	0.9449
Due date factor								
$\beta = 0.25$	1.000	1.2409	0.9323	1.0994	1.0777	0.9230	0.9276	0.9016
$\beta = 0.50$	1.000	2.4659	0.9302	1.6441	1.7911	1.0808	1.3071	0.9054
$\beta = 0.75$	1.000	3.8370	0.9231	1.9025	3.0732	1.2462	2.0104	0.9503
Overall	1.000	2.5146	0.9286	1.5487	1.9807	1.0833	1.4151	0.9191

Table 7
Effect of due-date factor and release date factor setting ($\Delta t = 4$; $thres = 10$)

Compare	BATC-II	BATC-I GA 1 BATC-I	BATC-II GA 1 BATC-II	BATC-III GA 1 BATC-III	GA 2 BATC-I —	GA 2 BATC-II —	GA 2 BATC-III —	GA 2 DTH —
$\alpha = 0.25, \beta = 0.25$	1.000	1.0063	0.9621	0.9934	0.9671	0.9134	0.9200	0.9285
$\alpha = 0.25, \beta = 0.50$	1.000	1.1540	0.9513	1.0708	1.1007	0.9195	0.9768	0.9255
$\alpha = 0.25, \beta = 0.75$	1.000	2.0381	0.9053	1.5738	1.9520	1.0401	1.4406	0.9486
$\alpha = 0.50, \beta = 0.25$	1.000	1.0997	0.9249	1.0592	1.0140	0.8625	0.8749	0.8594
$\alpha = 0.50, \beta = 0.50$	1.000	1.6107	0.9099	1.1794	1.2799	0.8600	0.9690	0.8352
$\alpha = 0.50, \beta = 0.75$	1.000	5.4066	0.8492	2.4557	3.8631	1.2363	2.2480	0.8317
$\alpha = 0.75, \beta = 0.25$	1.000	1.5656	0.9145	1.2172	1.2100	0.8991	0.9044	0.8826
$\alpha = 0.75, \beta = 0.50$	1.000	3.9075	0.8695	2.1534	2.1332	1.0836	1.4287	0.897
$\alpha = 0.75, \beta = 0.75$	1.000	1.7741	0.9769	1.2183	1.0791	0.9567	1.0277	0.9817

Here, we consider the fixed time window $\Delta t = 4$. Furthermore, we also fix the number of jobs used for batch formation $thres = 10$. We fix these values for the main experimentation because this provides a tradeoff between solution quality and time required for computation. We can see

Table 8
Effect of time window setting and *thres*-value setting

Compare	BATC-II	BATC-I GA 1 BATC-I	BATC-II GA 1 BATC-II	BATC-III GA 1 BATC-III	GA 2 BATC-I —	GA 2 BATC-II —	GA 2 BATC-III —	GA 2 DTH —
$\Delta t = 2, thres = 10$	1.000	2.1242	0.9490	1.5607	1.6144	1.0454	1.2758	0.9289
$\Delta t = 2, thres = 15$	1.000	2.1803	0.9554	1.9328	1.7406	0.9886	1.3958	0.8970
$\Delta t = 4, thres = 10$	1.000	2.1736	0.9182	1.4357	1.6221	0.9746	1.1989	0.8989
$\Delta t = 4, thres = 15$	1.000	1.9324	0.9257	1.7991	1.5532	0.9271	1.2769	0.9121
$\Delta t = 8, thres = 10$	1.000	2.2258	0.8888	1.3231	1.4096	0.8012	0.9948	0.7310
$\Delta t = 8, thres = 15$	1.000	2.0511	0.9109	1.6006	1.5542	0.8917	1.1392	0.8659

that all algorithms except BATC-I_GA-1_BATC-I lead to a TWT improvement for $\alpha = 0.25$ and $\beta = 0.25$.

The algorithm BATC-II_GA-1_BATC-II leads to larger improvements for small α values because in this case the arrival times of the jobs are clustered. A larger β value leads (for a fixed α value) to larger improvement rates. This is caused by the fact that a larger β value causes a wider spread of due dates. In this situation, the genetic algorithm performs better than the dispatching rule. An exception is the parameter combination $\alpha=0.75$ and $\beta=0.75$ that leads to a very small improvement. In this case, the ready times and the due dates of the jobs are both spread widely. Hence, there exists a lot of schedules with TWT close to zero and it is difficult for the GA to find better solutions.

The algorithms GA-2_BATC-II and GA-2_DTH realize the largest improvement rates for an average α value. Here, the improvement rate is almost independent from the used β value.

9.2. Effect of time window size (Δt) and number of jobs *thres*

In Table 8, we present the results for different time window sizes and different number of jobs (per family) *thres* used for batch formation. We conclude from Table 8 that a larger time window Δt leads to larger improvements. However, this is not true for the BATC-I_GA-1_BATC-I algorithm. An increasing value for *thres* usually decreases the improvement. This behavior is caused by the fact that an increasing *thres* value decreases the total weighted tardiness value obtained by the pure BATC-type rule significantly. Hence it is harder for the genetic algorithm-based approaches to find significantly better solutions.

9.3. Time required for computation

In Table 9, we find the time required for running BATC-II, GA 1, and GA 2 in seconds. All tests were performed on a Pentium 4, 1700 MHz Processor with 256 MB RAM. Table 9 indicates that the running time of the different algorithms are quite different.

Table 9
Computation time for the different algorithms ($\Delta t = 4$; $thres = 10$)

Compare	BATC-II	BATC-I GA 1 BATC-I	BATC-II GA 1 BATC-II	BATC-III GA 1 BATC-III	GA 2 BATC-I —	GA 2 BATC-II —	GA 2 BATC-III —	GA 2 DTH —
Machines								
$m = 3$	0.24	38.73	67.38	41.43	1060.34	1523.13	1016.70	5755.23
$m = 4$	0.24	34.71	58.18	36.51	851.25	1308.24	868.69	3541.50
$m = 5$	0.25	29.25	49.54	31.02	716.24	1117.88	712.12	2817.70
Batch size								
$B = 4$	0.22	58.01	98.19	61.32	1137.73	1677.85	1114.23	3911.37
$B = 8$	0.26	10.46	18.55	11.32	614.83	955.95	617.45	4164.92
Release time factor								
$\alpha = 0.25$	0.28	40.27	77.78	43.31	1221.76	1891.42	1202.51	6573.10
$\alpha = 0.50$	0.25	34.06	60.84	36.71	859.94	1343.43	859.50	3995.15
$\alpha = 0.75$	0.21	28.35	36.48	28.95	547.13	715.86	535.50	1546.19
Due date factor								
$\beta = 0.25$	0.23	33.31	57.55	34.03	887.43	1351.08	871.91	4029.56
$\beta = 0.50$	0.25	34.34	62.82	37.00	885.45	1372.46	889.18	4246.22
$\beta = 0.75$	0.25	35.04	54.72	37.93	855.96	1227.18	836.42	3838.67
Overall	0.24	34.23	58.37	36.32	875.95	1316.42	865.84	4038.15

If a quick solution is required, then BATC-II should be used independently. Better solutions with respect to TWT can be found by using GA 1 and GA 2. GA 1 is much faster than GA 2 because it takes more time to form the batches. In GA 2, in each generation for each element of the population, a new batch formation step is required. Furthermore, the number of generations used is much higher in case of GA 2.

The computation time of the different algorithms also strongly depends on the batch size. In the case of a large batch size, there exist fewer possibilities to form batches. Hence, the time required for batch formation is smaller. We can see this effect even for GA 2. Another factor for the computation time is the range of the ready times. In the case of widely spread ready times, we have a greater clustering of jobs at each batch formation step. Hence, we have to consider more batch combinations and this increases the time required for computation. The algorithm GA-2_DTH results in very long computing times. This is caused by the fact that at each point of decision, a large number of computations have to be performed in order to consider the possible batches formed.

9.4. Influence of the number of incompatible families on the solution quality

We do not include the number of families in our experimental design in order to reduce the number of test instances, which are computationally expensive. However, it seems worthwhile to investigate the impact of the number of incompatible families on the solution quality. Therefore,

Table 10
Solution quality for different number of families ($\Delta t = 4$; $thres = 10$)

Compare	BATC-II	BATC-I GA 1 BATC-I	BATC-II GA 1 BATC-II	BATC-III GA 1 BATC-III	GA 2 BATC-I —	GA 2 BATC-II —	GA 2 BATC-III —
$f = 3$	1.0000	1.6378	0.9442	1.3633	1.6974	0.9743	1.5863
$f = 6$	1.0000	1.0790	0.9634	1.0242	1.1429	0.9974	0.9922
$f = 12$	1.0000	0.9813	0.9708	0.9754	1.0321	1.0136	1.0115

we run experimental cases with three, six and 12 incompatible families ($f = 3, 6$ and 12). We use four machines, a batch size of four and $\alpha = 0.50$, $\beta = 0.50$. The total number of jobs to be scheduled for all the test cases is fixed at 180 and the number of jobs per family is $180/f$. The results of the experiments are presented in Table 10, and show that the improvements obtained by BATC-II_GA-1_BATC-II are the highest for three-family case. For the 12-family case, however, the solution qualities of all the approaches are close. Therefore, we conclude that it is beneficial to use BATC-II_GA-1_BATC-II when few product families exist and the BATC-II rule when the number of families is high.

9.5. Approach 1 vs Approach 2

Approach 1 forms batches, then assigns the batches to machines and finally sequences them on each machine; while approach 2 assigns jobs to machines, then forms batches and finally sequences them on each machine. Batches are fixed in the first stage in approach 1 and so in comparison, approach 2 has more opportunities to explore good assignments and batch formation with respect to TWT. On the other hand, the batch formation before the assignment step in approach 1 reduces the problem size dramatically. In approach 2, we have to form batches for each generation and each element of the population. Hence, its computational time is much higher. Because of the larger search space for approach 2, the convergence of the genetic algorithm is slow. Furthermore, there is a greater risk of getting stuck in local optima than approach 1. Fig. 3 shows a graphical comparison of the computation times of GA 1 and GA 2 algorithms based on the levels of the different factors (computation times that are very similar overlap to a single point on the graph). It is clear that GA 2 algorithms are significantly slower and that the GA 1 algorithms are much faster.

In summary, approach 1 generally produces better results with respect to solution quality and time required for computing. Only the combination GA-2_DTH produces better results with respect to TWT. However, the computation time required for GA-2_DTH is much higher (see Table 9).

10. Conclusions and future work

In this paper, we investigated two different approaches for scheduling jobs with incompatible job families and unequal ready times on parallel batch machines. The first approach forms batches first, then assigns the batches to individual machines using a genetic algorithm, and finally sequences them.

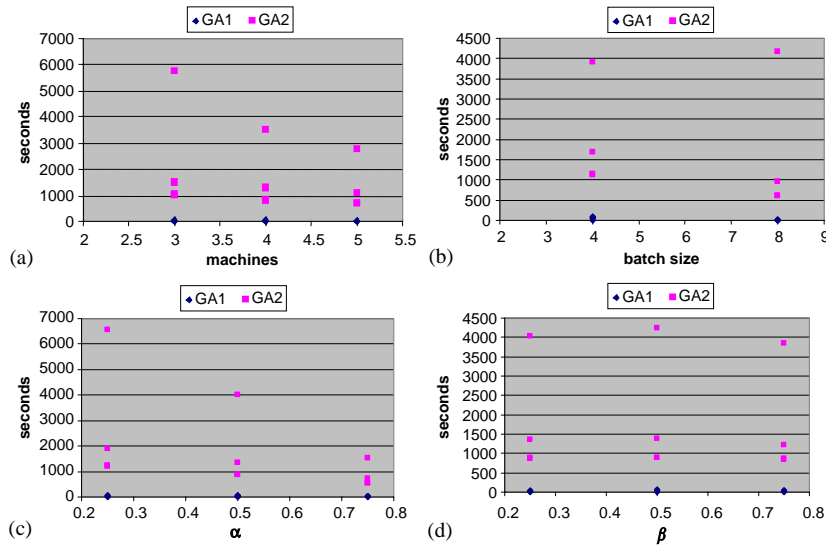


Fig. 3. Graphs for the computation time for the different algorithms ($\Delta t = 4$; $thres = 10$).

The second approach first assigns jobs to individual machines and then forms the batches before sequencing them. We use time window techniques for forming the batches. For sequencing of the batches, we consider modifications of the apparent tardiness cost dispatching rule. In the case of the second approach, we also implemented a modification of the decision theory approach of Kanet and Zhou [17] for simultaneous batch formation and sequencing. By using stochastically generated test data, we conclude that approach 1 usually outperforms: (1) all except one (GA2-DTH) algorithm of approach 2 with respect to solution quality and (2) all algorithms of approach 2 with respect to computation time.

The derivation of local dominance rules for the batching case by extending the work of Akturk and Ozdemir [26] appears as an important part of future research. Some promising first steps into this direction are reported by Kurz [30]. Research is also required in determining lower bounds to evaluate how far the solutions obtained by heuristic techniques are away from the optimal solution.

Acknowledgements

The authors gratefully acknowledge the support of International SEMATECH and the Semiconductor Research Corporation through the Factory Operations Research Center (FORCe) project 2001-NJ-880. The authors would like to thank Jeaninne Schmidt for fruitful discussions and her valuable programming and testing efforts. Parts of this research were carried out while the first author was visiting the Modeling and Analysis of Semiconductor Manufacturing (MASM) Laboratory at Arizona State University.

References

- [1] Mehta SV, Uzsoy R. Minimizing total tardiness on a batch processing machine with incompatible job families. *IIE Transactions* 1998;30:165–78.
- [2] Lawler EL. A “pseudopolynomial” time algorithm for sequencing jobs to minimize total weighted tardiness. *Annals of Discrete Mathematics* 1977;1:331–42.
- [3] Perez TI, Fowler JW, Carlyle WM. Minimizing total weighted tardiness on a single batch process machine with incompatible job families. *Computers and Operations Research*, to appear. DOI: [10.1016/S0305-0548\(03\)00239-9](https://doi.org/10.1016/S0305-0548(03)00239-9)
- [4] Azizoglu M, Webster S. Scheduling a batch processing machine with incompatible job families. *Computers and Industrial Engineering* 2001;39(3–4):325–35.
- [5] Graham RL, Lawler EL, Lenstra JK, Rinnooy Kann AHG. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* 1979;5:287–326.
- [6] Dupont L, Dhaenens-Flipo C. Minimizing the makespan on a batch machine with non-identical job sizes: an exact procedure. *Computers and Operations Research* 2002;29(7):807–19.
- [7] Wang C, Uzsoy R. A genetic algorithm to minimize maximum lateness on a batch processing machine. *Computers & Operations Research* 2002;29(12):1621–40.
- [8] Devpura A, Fowler JW, Carlyle M, Perez I. Minimizing total weighted tardiness on a single batch processing machine with incompatible job families. *Proceedings of the Symposium on Operations Research 2000*. p. 366–71.
- [9] Balasubramanian H, Mönch L, Fowler JW, Pfund ME. Genetic algorithm based scheduling of parallel batch machines with incompatible families to minimize total weighted tardiness. *International Journal of Production Research* 2004;42(8):1621–38.
- [10] Mönch L, Balasubramanian H, Fowler JW, Pfund ME. Minimizing total weighted tardiness on parallel batch processing machines using genetic algorithms. *Proceedings of the International Symposium on Operations Research, Klagenfurt, Austria; 2002*. p. 205–11.
- [11] Glassey CR, Weng WW. Dynamic batching heuristics for simultaneous processing. *IEEE Transactions on Semiconductor Manufacturing* 1991;4(2):77–82.
- [12] Fowler JW, Phillips DT, Hogg GL. Real-time control of multiproduct bulk-service semiconductor manufacturing processes. *IEEE Transactions on Semiconductor Manufacturing* 1992;5(2):158–63.
- [13] Fowler JW, Hogg GL, Phillips DT. Control of multi-product bulk-server diffusion/oxidation processes part two: multiple servers. *IIE Transactions on Scheduling and Logistics* 2000;32(2):167–76.
- [14] Robinson JK, Fowler JW, Bard JF. The use of upstream and downstream information in scheduling semiconductor batch operations. *International Journal of Production Research* 1995;33(7):1849–69.
- [15] Cigolini R, Perona M, Portioli A, Zambeli T. A new dynamic look-ahead scheduling procedure for batching machines. *Journal of Scheduling* 2002;5:185–204.
- [16] Duenyas I, Neale JJ. Stochastic scheduling of a batch processing machine with incompatible job families. *Annals of Operations Research* 1997;70:191–220.
- [17] Chand S, Traub R, Uzsoy R. Rolling horizon procedures for the single machine deterministic total completion time scheduling problem with release dates. *Annals of Operations Research* 1997;70:115–25.
- [18] Mason SJ, Fowler JW, Carlyle WM. A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops. *Journal of Scheduling* 2002;5(3):247–62.
- [19] Dimipoulos C, Zalzalá AMS. Recent development in evolutionary computation for manufacturing optimization: problems, solutions and comparisons. *IEEE Transactions on Evolutionary Computation* 2000;4(2):93–113.
- [20] Aytug H, Khouja M, Vergara FE. Use of genetic algorithms to solve production and operations management problems: a review. *International Journal of Production Research* 2003;41(17):3955–4009.
- [21] Gravel M, Price WL, Gagne C. Scheduling jobs in an alcan aluminium foundry using a genetic algorithm. *International Journal of Production Research* 2000;38(13):3031–41.
- [22] Fowler JW, Horng S, Cochran JK. A hybridized genetic algorithm to solve parallel machine scheduling problems with sequence dependent setups. *International Journal of Industrial Engineering* 2003;10(3):232–43.
- [23] Vepsäläinen APJ, Morton TE. Priority rules for job shops with weighted tardiness costs. *Management Science* 1987;33(8):1035–47.
- [24] Ovacik IM, Uzsoy R. Rolling horizon procedures for dynamic parallel machine scheduling with sequence dependent setup times. *International Journal of Production Research* 1995;33:3173–292.

- [25] Morton TE, Ramnath P. Guided forward search in tardiness scheduling of large one machine problems. In: Brown DE, Scherer WT, editors. *Intelligent scheduling systems*. Hingham, MA: Kluwer Academic Publishers; 1995.
- [26] Akturk MS, Ozdemir D. A new dominance rule to minimize total weighted tardiness with unequal release dates. *European Journal of Operational Research* 2001;135:394–412.
- [27] Kanet JJ, Zhou Z. A decision theory approach to priority dispatching for job shop scheduling. *Production and Operations Management* 1993;2(1):2–13.
- [28] Michalewicz Z. *Genetic algorithms+Data structures=Evolution algorithms*, 3rd ed. Berlin, Heidelberg, New York: Springer; 1996.
- [29] Wall M. “Galib: A C++ library of genetic algorithms components” Website: <http://lancet.mit.edu/ga/>, 1999.
- [30] Kurz ME. On the structure of optimal schedules for minimizing total weighted tardiness on parallel batch-processing machines. *Proceedings of the 10th Industrial Engineering Research Conference*, Portland, Oregon; 2003.