

Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness

HARI BALASUBRAMANIAN[†], LARS MÖNCH[‡], JOHN FOWLER[†] and MICHELE PFUND[†]

This research is motivated by a scheduling problem found in the diffusion and oxidation areas of semiconductor wafer fabrication, where the machines can be modelled as parallel batch processors. We attempt to minimize total weighted tardiness on parallel batch machines with incompatible job families. Given that the problem is NP-hard, we propose two different versions of a genetic algorithm (GA), each consisting of three different phases. The first version forms fixed batches, then assigns batches to the machines using a GA, and finally sequences the batches on individual machines. The second version assigns jobs to machines using a GA, then forms batches on each machine for the jobs assigned to it, and finally sequences these batches. Heuristics are used for the batching phase and the sequencing phase. For both these versions an additional fourth phase can be included wherein the sequenced batches are modified using pairwise swapping techniques. Using stochastically generated test data we show that algorithms of the first version of the GA outperform (1) traditional dispatching rules with respect to solution quality and (2) the algorithms of the second version with respect to both solution quality and computation time.

1. Introduction

Semiconductor wafer fabrication consists of hundreds of process steps and the complexity is increased by re-entrant product flows, sequence-dependent setups, diversity of product mix and batch-processing machines. Such a complex manufacturing environment makes it difficult to meet customer due dates and ensure on-time delivery. In this research we focus on diffusion and oxidation operations which are performed in batch processing machines, i.e. machines where several jobs can be processed together simultaneously. Some batch processing steps in the semiconductor industry have extremely long processing times (10 hours) compared to other operations (1–2 hours). Because of these long processing times, effective scheduling of these operations is important to achieving good system performance (Mehta and Uzsoy 1998). While several jobs can be processed at the same time, jobs of different families cannot be processed together due to the chemical nature of the process. Furthermore, depending on customer requirements, some jobs have a higher priority index than others and products need to meet internal/external due dates.

In this research, diffusion and oxidation machines are modelled as parallel batch processing machines with incompatible job families, and the performance measure to

Revision received July 2003.

[†]Department of Industrial Engineering, Arizona State University, Tempe, AZ, 85287-5906.

[‡]Department of Information Systems, Institut of Information Systems, Technical University of Ilmenau, Ilmenau, Germany, PSF 100 565, D-98684.

*To whom correspondence should be addressed. e-mail: john.fowler@asu.edu

be minimized is total weighted tardiness. Total weighted tardiness is the summation of the weighted tardiness over all jobs $j = 1, 2, 3, \dots, n$, i.e. $\sum w_j T_j$ where $T_j = \max(0, C_j - d_j)$, w_j is the weight (priority), C_j is the completion time and d_j the due date of job j . This problem, referred to as $Pm | \text{batch, incompatible} | \sum w_j T_j$ is NP-hard (Pinedo 1995, by reduction to $1 || \sum w_j T_j$ which is NP-hard by Lawler 1977). This research proposes heuristic approaches that lead to good solutions.

The paper is organized as follows. In section 2 we discuss previously done research involving the scheduling of batch machines. We define the problem and assumptions in section 3. In section 4, we explain the decomposition approaches proposed and the details of the heuristics used in this study, including the genetic algorithm; we also describe the parameter settings of the GA and the test data generation scheme. In section 5, we present the results from the computational experiments. Finally, section 6 contains our conclusions and future research.

2. Previous related work

Many researchers have addressed problems related to batching machines. However, the various problems investigated have been for different machine environments, different constraints, and different performance measures. Perez (1999) provides a detailed review and classification of papers that have dealt with deterministic scheduling problems related to batching. Azizoglu and Webster (2001) use a branch-and-bound procedure to schedule a single batch-processing machine with incompatible job families where the jobs have arbitrary sizes and weights to minimize total weighted completion time ($1 | \text{batch, non-identical, incompatible} | \sum w_j C_j$). Dupont and Dhaenens-Flipo (2002) consider the problem of a single batch-processing machine in which jobs have non-identical capacity requirements to minimize the makespan ($1 | \text{batch, non-identical} | C_{\max}$). Using dominance rules they develop a branch-and-bound procedure for this problem. Wang and Uzsoy (2002) consider the problem of scheduling a single batch machine with job release dates to minimize maximum lateness ($1 | \text{batch, } r_j | L_{\max}$). They use a genetic algorithm coupled with dynamic programming techniques to solve the problem.

The references most closely related with the scheduling problem being researched herein are Mehta and Uzsoy (1998), Perez (1999) and Devpura *et al.* (2000). All three deal with the single-machine batching problem with incompatible job families. Mehta and Uzsoy (1998) consider the objective of minimizing total tardiness ($1 | \text{batch, incompatible} | \sum T_j$), and suggest a decomposition approach. The problem is decomposed into two parts: batching the jobs, and sequencing the already batched jobs. Perez (1999) extends this work to minimizing total weighted tardiness ($1 | \text{batch, incompatible} | \sum w_j T_j$), and the same set of heuristics is modified to include weights. Since the batches are fixed when the sequencing is done, the potential improvement lies only in the sequencing phase. Devpura *et al.* (2000) suggest a pairwise interchange heuristic that changes the composition of batches and produces considerable improvement.

In this paper we extend the single-machine batching problem considered by Perez (1999) and Devpura *et al.* (2000) to parallel machines.

3. Problem definition and assumptions

Using the $\alpha | \beta | \gamma$ notation for scheduling (Pinedo 1995) this problem can be represented as $P_m | \text{batch, incompatible} | \sum w_j T_j$. Our assumptions are

as follows:

1. Jobs of the same family have the same processing times.
2. All jobs are available at time zero.
3. All the batch-processing machines are identical in nature.
4. Once a batch-processing machine is started, it cannot be interrupted. No preemption 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 waiting 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 batch processing machine capacity is B jobs.
10. Batch k of family j is represented by B_{kj} .
11. The completion time of job i of family j is denoted as C_{ij} .
12. The weighted tardiness of job ij is represented as $w_{ij}T_{ij} = w_{ij} \cdot \max \{C_{ij} - d_{ij}, 0\}$.

4. Decomposition approaches

4.1. Description

Due to the complexity of the problem two different three-stage decomposition approaches are investigated. The first decomposition approach (Approach 1) forms the batches in the first stage, assigns the batches to the machines in the second stage, and schedules the batches on each of the machines in the third stage. The stages are shown in figure 1 using an eight-job, two-family, and two-machine example with batch size 2.

The second decomposition approach (Approach 2) assigns the jobs to the machines in the first stage, forms batches on each machine in the second stage,

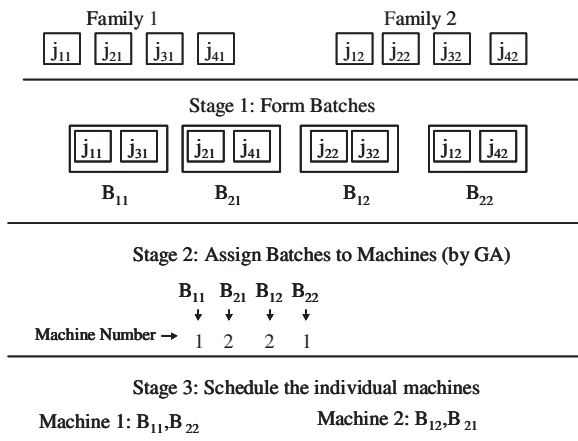


Figure 1. Approach 1.

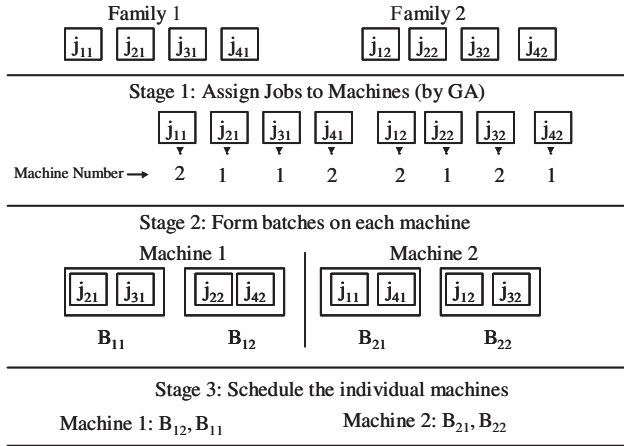


Figure 2. Approach 2.

and schedules these batches on their respective machines in the third stage. The three stages of this approach for the same example are shown in figure 2.

4.2. The use of genetic algorithms

GAs have been used extensively in combinatorial optimization problems including scheduling problems. GAs applied to scheduling problems consider schedules as individuals of a population. A fitness value derived from the objective value of the schedule is assigned to each single individual, called a chromosome. GAs work iteratively. A single iteration is called a generation. The individuals of the new generation are obtained from the individuals of the previous one by applying reproduction and mutation procedures. Only the more fit individuals are selected.

The references most pertinent to this research paper are by Gravel *et al.* (2000) and Fowler *et al.* (2001). Gravel *et al.* (2000) use a double-loop genetic algorithm for a parallel machine scheduling problem within an aluminium foundry under special process restrictions. Fowler *et al.* (2001) applied genetic algorithms to solve scheduling problems involving parallel machines with sequence-dependent setups. They propose a hybridized genetic algorithm that also uses traditional dispatching rules. The GA is used to assign jobs to machines, and after this assignment is done, the single-machine sub-problems are solved.

Similarly, in both of the suggested approaches in this research (Approach 1 and Approach 2), the GA is used for the assignment of jobs or batches to machines. In Approach 1, the GA assigns batches to machines, while in Approach 2, the GA assigns jobs to the 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 the dispatching and scheduling rules) on all the machines. Hopefully the GA converges towards assignments that give good solutions with respect to total weighted tardiness. The GA is used for assignment purposes only; it is hybridized with heuristic approaches for batching and sequencing as suggested in Perez (1999). Thus two different versions of the GA are developed: GA version 1 (based on Approach 1) and GA version 2 (based on Approach 2). The approaches are summarized in figures 3 and 4.

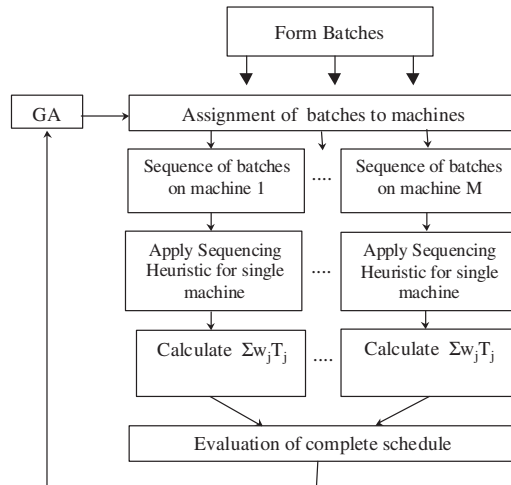


Figure 3. GA version 1.

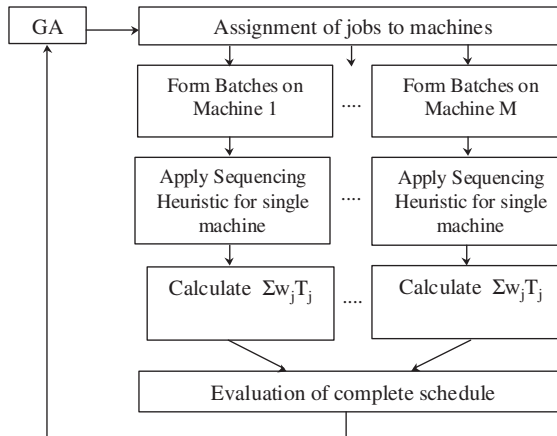


Figure 4. GA version 2.

4.3. Heuristics for batching and sequencing

4.3.1. The ATC (apparent tardiness cost) dispatching rule

The well-known ATC (apparent tardiness cost) rule suggested by Vepsalainen and Morton (1987) plays an important role in this research.

1. *ATC used independently.* This approach does not consider the assignment of jobs or batches to the machines at the outset, and therefore does not require the GA. Instead every time a machine becomes free, one batch from each family is formed, and of all the formed batches (the number of batches will be equal to the number of families) one is chosen and scheduled on the machine. This heuristic is simple but has been known to give reasonable solutions quickly (Perez 1999, Devpura *et al.* 2000).

The ATC index I_{ij} for a job i belonging to family j calculated at time t is given below:

$$I_{ij}(t) = \left(\frac{w_{ij}}{p_j} \right) \exp \left(\frac{-\max(d_{ij} - p_j - t, 0)}{k\bar{p}} \right).$$

In the index, k is the look-ahead parameter, \bar{p} is the average processing time of the remaining unscheduled jobs, and t is the time of the scheduling decision.

The jobs in each family are ordered in decreasing order of their ATC indices, and a batch of the first B jobs is formed per family. In order to schedule one of these batches the batched apparent tardiness cost (BATC) rule is used: at time t , for all the batches the following index is calculated

$$BATC_{xj} = \sum_{i \in B_{xj}} I_{ij}(t)$$

where $BATC_{xj}$ is the BATC index for batch x of family j . The batch with the highest BATC index is scheduled. Thus, the BATC is a sequencing rule for batches that have been formed.

One of the objectives of this research is to evaluate the performance of the ATC-BATC rule (used independently) with the more sophisticated approaches using the GA, and understand the trade-offs between solution quality and computation time.

2. *Using ATC for forming batches only.* When the ATC heuristic is used independently we have a complete schedule of jobs in batches and batches sequenced on machines. However, this heuristic could be used solely for batching if we consider the machines unscheduled, and use only the batches. These batches have been formed dynamically (not at time $t=0$), and therefore is an effective way of batching the jobs. This technique of batching is used in GA version 1, which requires batches to be formed in the first stage. The steps are described below:

Run ATC-BATC independently → Use only the batches formed → Assign the batches to machines using the GA → Sequence the batches on their respective machines.

In GA version 2, the ATC rule is used to batch the jobs only after they have been assigned to the machines. The procedure is the same as that used by Perez (1999). The steps are described below:

Assign jobs to machines → On each machine batch jobs by ATC → Schedule the batches on their respective machines.

As the ATC-based 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 1998). For a particular test case, we run the ATC-BATC independently for each k -value, calculate the total weighted tardiness value, and choose the k -value that gives the least total weighted tardiness. This k -value chosen was then fixed (for a given test case) for GA version 1 and GA version 2 algorithms that use ATC-based rules.

4.3.2. Other heuristics

Several other heuristics were suggested by Perez (1999) apart from the ATC-based rules:

1. EDD (earliest due date): The EDD rule can be used for both batching and sequencing. For batching the EDD heuristic orders the jobs in descending order of due dates for each family and then forms full batches starting with the smallest due date. For sequencing, each batch is assigned a due date equal to the due date of the job that is earliest in the batch. Batches are sequenced in the descending order of these due dates. This method completely discounts the weights or priorities of the

jobs; Perez (1999) and Devpura *et al.* (2000) have reported on its poor performance when compared to other heuristics, but we include it for comparison purposes.

2. DH (decomposition heuristic): This approach is used for single-machine batch sequencing only and is similar to the one described in Mehta and Uzsoy (1998) and extended by Perez (1999). From an initial sequence obtained by a heuristic (e.g. BATC), a subsequence of size λ is solved into an optimal sequence using complete enumeration. The first α batches of this optimal subsequence are fixed into the final sequence, and the next λ unscheduled batches are considered. This process is repeated until all jobs are scheduled and no improvement is made in the total weighted tardiness of the final sequence. The larger the value of λ , the higher the computation time and the better the solution quality. However, the GA considers a number of solutions in each generation, which leads to high computation times if the DH procedure is used. In order to keep the computation time of the GA reasonable we experimented with different values of λ and α , and set the values $\lambda = 5$ and $\alpha = 2$.

4.4. Corrective techniques (swapping)

To improve upon the solution obtained by the two decomposition approaches, a corrective fourth stage where the composition of batches is changed can be included. After we get a complete solution of batches sequenced on all the machines, we attempt to improve the solution through swapping techniques suggested by Devpura *et al.* (2000).

We propose an extension of the Devpura *et al.* (2000) swapping for 1 | batch, incompatible | $\sum w_j T_j$ to the parallel machine case. When parallel machines are considered, the jobs can be redistributed not only across compatible batches of the same family on the same machine, but also across machines.

We implement the swapping algorithm as follows. We keep the position of the batches fixed and try to interchange jobs between batches of the same family. Initially, for each family, we consider the batch with the earliest start time. Considering all jobs in the batch starting from first to last, we look for the possibility of a swap (better solution) with jobs in other compatible batches with start times later than it, in the order of non-decreasing start times. Whenever a swap occurs we go back to the first job of the batch and start over. When there are no more improvements in this batch we move on to the next batch and check if we can swap the jobs in it with jobs in batches scheduled after it. The procedure is continued till all the batches of all the families are covered. The procedure explained above is for two-way swaps; Devpura *et al.* (2000) show that all cases in a three-way swap are covered with the rules for two-way swaps.

4.5. Implementation of the GA

As stated in section 4.2, the GA is used in the assignment phase of the two decomposition approaches (Approach 1 and Approach 2). In Approach 1, the GA is used to assign batches to machines while in Approach 2 the GA is used to assign jobs to machines. A single chromosome represents a particular assignment of jobs or batches to machines. Each chromosome has an objective value (total weighted tardiness) that is equal to the sum of the objective values obtained of each of the machines after they have been sequenced. The following steps describe in detail how the GA is used in this research: coding, initialization, selection, crossover, mutation, evaluation

and termination. Since the GA can be used to assign batches or jobs, the terms entity or entities are used to represent both.

1. *Coding*: A solution to the problem of assigning entities to machines is an array whose length is equal to the number of entities to be scheduled. Each element represents the machine to which the entity is assigned. For example, consider a problem with six entities to be assigned to three machines. A feasible solution, represented as an array of length 6, could be [2 2 3 1 1 3]. Thus the first entity has been assigned to machine 2, the second also to machine 2, the third to machine 3 and so on. The completed assignment is shown below:

Machine 1: Entity 4 and Entity 5
 Machine 2: Entity 1 and Entity 2
 Machine 3: Entity 3 and Entity 6

In the terminology of genetic algorithms, the array is called the chromosome and each element is called a gene.

2. *Initialization*: Two different initialization schemes (called Scheme 1 and Scheme 2) are proposed. Scheme 1 involves random initialization and can be used in both Approach 1 and Approach 2. Each chromosome is randomly initialized by generating numbers from 1 to the total number of machines (m) available for each of the n entities to be assigned. Scheme 1 of initialization can be described as follows:

1. Set $i = 1$.
2. Generate an integer random variate r between 1 and m .
3. Assign entity i to machine r .
4. Set $i = i + 1$. If $i > n$ STOP, else go to step 2.

Scheme 2 is only applicable when the GA assigns batches to the machines (Approach 1). The assignment of batches to machines generated by the well-known ATC dispatching rule (Vepsalanien and Morton 1987) is perturbed to generate a certain percentage of the initial population.

The steps are:

1. Generate an initial solution using the ATC-BATC dispatching rule independently.
2. Identify the assignments of batches to machines from this solution.
3. Randomly pick two batches b_1 and b_2 and whose assignments by ATC are m_1 and m_2 i.e. $b_1 \rightarrow m_1$ and $b_2 \rightarrow m_2$.
4. Interchange the machine assignments, i.e. $b_1 \rightarrow m_2$ and $b_2 \rightarrow m_1$.
5. Repeat steps 1–4 for a specified percentage p of the total population size.

3. *Selection*: For selection of chromosomes for reproduction the method followed is the roulette wheel technique (Michalewicz 1996, Wall 1999). The fitness of the chromosome is the total weighted tardiness value. The probability of selection of a certain chromosome is proportional to its fitness; the fitness is calculated using linear scaling.

4. *Crossover*: A one-point crossover (Michalewicz 1996) is utilized in which two parent chromosomes are selected randomly according to fitness, and the crossover point is randomly chosen.

5. *Mutation*: Each gene of each of the resulting offspring is randomly changed to a different machine number (flip operator) with a predefined 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 fraction of the population that is replaced is set with a predefined probability called the replacement probability.

7. *Stopping criterion:* The genetic algorithm is controlled by a specified 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. If either of these two stopping criteria is satisfied, the GA is stopped.

The GA is implemented based on the object-oriented C++ framework Galib (Wall 1999) that supports the chromosome representations and the steps of the GA. The steps of the GA are similar to those described by Mönch (2002).

4.6. Algorithms

A number of algorithms are available with which both versions of the GA can be hybridized. For batching, EDD and ATC heuristics can be used; for sequencing, EDD, BATC and DH can be used. In addition, after sequencing, it is possible to include swapping between batches of compatible families. Also, for GA version 1 it is possible to have two different initialization schemes: random initialization and initialization based on ATC. Several hybrid algorithms are proposed using a combination of the available heuristics. All the proposed algorithms are listed in table 1. If the algorithm belongs to GA version 1, the assignment stage is written as GA1; and if the algorithm belongs to GA version 2, the assignment stage is written as GA2. To differentiate between the initialization schemes on GA1, GA1a is used for initialization scheme 1 and GA1b is used for initialization scheme 2.

Apart from these twelve different algorithms, the ATC-BATC rule used independently is also considered. The ATC-BATC rule takes only a few seconds to generate a solution while the GA takes much longer. In the results section (section 2.8), all the algorithms based on the two versions of the GA are compared with the ATC-BATC rule. We also include the ATC-BATC-swap rule (without the GA) to check the improvements that swapping can yield over ATC-BATC.

Alg. No.	Stage 1	Stage 2	Stage 3	Stage 4
1	EDD	GA1a	EDD	-
2	ATC	GA1a	BATC	-
3	ATC	GA1a	DH	-
4	ATC	GA1a	BATC	Swap
5	ATC	GA1a	DH	Swap
6	ATC	GA1b	BATC	Swap
7	ATC	GA1b	DH	Swap
8	GA2	EDD	EDD	-
9	GA2	ATC	BATC	-
10	GA2	ATC	DH	-
11	GA2	ATC	BATC	Swap
12	GA2	ATC	DH	Swap

Table 1. Algorithms.

4.7. Parameter settings for the GA

The performance of a GA is generally sensitive to the settings of the parameters that influence the search behaviour of the algorithm, and our research is not an exception to this. It is highly desirable, therefore, to set these parameters to levels that will produce high-quality solutions. In order to arrive at the parameters for both the GAs we used a response surface methodology as described in Myers and Montgomery (2002). Preliminary experiments revealed that there were three important factors: population size, crossover probability and mutation probability. We represent the factors with their low and high values as follows: Factor (Low, High).

1. Population size (10,150)
2. Crossover probability (0.5, 0.9)
3. Mutation probability (0.0001, 0.01).

We used a 2^3 factorial design and 10 runs at each factor-level combination. The designed experiment was conducted separately for both the versions of the GA. After determining the regression equation, we performed the steepest descent technique to identify the region of improved response. The final parameter values used for the two versions of the GA are listed in tables 2 and 3.

4.8. Test data generation

For testing our heuristic approaches, random problem instances were generated in a manner similar to Mehta and Uzsoy (1998) and subsequently adapted by Perez (1999) and Devpura *et al.* (2000). Since we are dealing with parallel batch machines, we include the number of machines as an added factor. Table 4 shows the factors used for test generation and their different levels. For due-date generation, there are two parameters: R and T . The value of R controls the range of the due dates, while the value of T represents the approximate percentage of tardy jobs.

Parameter	Value
Population size	133
Crossover probability	0.8
Mutation probability	0.009
Replacement probability	0.6
Diversity	0.1
Number of generations	80

Table 2. Parameter settings for GA version 1.

Parameter	Value
Population size	115
Crossover probability	0.8
Mutation probability	0.01
Replacement probability	0.6
Diversity	0.1
Number of generations	200

Table 3. Parameter settings for GA version 2.

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
	2 with a probability of 0.2	
	4 with a probability of 0.2	
Family processing time	10 with a probability of 0.3	
	16 with a probability of 0.2	
	20 with a probability of 0.1	
Weight per job w_{ij}	Uniform (0,1)	1
	$d_{ij} = \text{Uniform} [\mu - (\mu R/2), \mu + (\mu R/2)]$	
	$\mu = \text{makespan} \times (1 - T)$	
Job due dates, d_{ij}	$\text{makespan} = n_j \times E[P] / (m \times B)$	
	$E[P] = \text{Expected processing time}$	
	$T = 0.3, 0.6 \quad R = 0.5, 2.5$	
	Total parameter combinations	72
	Number of problems/combinations	10
	Total Problems	720

Table 4. Parameters and their different levels.

5. Results

The results are summarized in tables 5 and 6. They have been presented as follows:

1. Instead of comparing all cases individually, the cases were grouped according to factor levels such as machines, batch size, etc. For example, results for a batch size of 4 imply that all other factors have been varied, but the batch size has been kept constant at 4.
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 ATC-BATC rule (used independently without the GA).
3. In addition to average case solution ratios, the worst case solution ratios for the algorithms are shown in table 6. For a particular algorithm, the worst case solution ratio indicates the worst performance of the algorithm over all the test cases considered under the level of a certain factor.

5.1. Overview of results

It is clear that the ATC-BATC rule is improved upon by several algorithms. The most effective algorithms are ATC-GA1a-DH-Swap and ATC-GA1b-DH-Swap; both algorithms are part of GA version 1. ATC-GA1b-BATC-Swap and ATC-GA1b-BATC-Swap perform almost as well, and are also a part of GA version 1. These algorithms give an average improvement of almost 10%. However the improvements for some levels of factors (for instance $R=2.5$ and $T=60\%$) are in the range of 3–5%, while at other levels ($R=0.5$ and $T=30\%$), the improvements are in the range of 12–15%. This indicates that the performance of the ATC-BATC is influenced by the due-date range factor R and the number of tardy jobs T . An analysis explaining the effect of R and T on the ATC-BATC is given in section 5.2.

Compare	ATC BATC	ATC BATC swap	EDD GA1 EDD	ATC GA1a BATC	ATC GA1a DH	ATC GA1a BATC swap	ATC GA1a DH swap	ATC GA1b BATC swap	ATC GA1b DH swap	GA2 EDD EDD	GA2 ATC BATC	GA2 ATC DH	GA2 ATC BATC swap	GA2 ATC DH Swap
Machines														
m = 3	1	0.96	1.96	0.99	0.97	0.94	0.90	0.92	0.91	1.74	1.27	1.26	0.96	0.96
m = 4	1	0.98	1.96	0.98	0.98	0.92	0.89	0.92	0.91	1.69	1.25	1.23	0.94	0.94
m = 5	1	0.96	1.86	0.98	0.98	0.92	0.92	0.93	0.91	1.62	1.23	1.21	0.93	0.92
Batch Size														
B = 4	1	0.97	2.04	0.99	0.98	0.93	0.92	0.93	0.92	1.80	1.29	1.25	0.95	0.95
B = 8	1	0.97	1.81	0.97	0.97	0.92	0.89	0.91	0.91	1.56	1.22	1.22	0.93	0.93
Jobs/Family														
60	1	0.95	1.86	0.96	0.95	0.93	0.89	0.91	0.89	1.59	1.20	1.18	0.93	0.93
80	1	0.98	1.92	1.00	0.98	0.92	0.91	0.92	0.91	1.69	1.28	1.25	0.95	0.94
100	1	0.97	2.01	0.99	0.99	0.92	0.91	0.93	0.93	1.77	1.28	1.27	0.95	0.95
Due-Date Variance														
R = 0.5	1	0.95	2.38	0.96	0.95	0.87	0.85	0.88	0.86	1.93	1.16	1.14	0.86	0.85
R = 2.5	1	0.99	1.47	1.00	1.01	0.98	0.96	0.97	0.97	1.44	1.35	1.33	1.03	1.03
% of Tardy Jobs														
T = 30%	1	0.95	1.96	0.97	0.96	0.88	0.86	0.88	0.87	1.58	1.32	1.29	0.89	0.89
T = 60%	1	0.99	1.89	1.00	1.00	0.97	0.95	0.96	0.95	1.79	1.19	1.18	1.00	0.99
Overall	1	0.96	1.93	0.98	0.98	0.92	0.90	0.92	0.91	1.68	1.25	1.23	0.94	0.94

Table 5. Average ratio of performance of specified heuristic to performance of ATC–BATC.

Compare	ATC BATC	ATC BATC swap	EDD GA1 EDD	ATC GA1a BATC	ATC GA1a DH	ATC GA1a BATC swap	ATC GA1a DH swap	ATC GA1b BATC swap	ATC GA1b DH swap	GA2 EDD	GA2 ATC BATC	GA2 ATC DH	GA2 ATC BATC swap	GA2 ATC DH Swap
Machines														
m = 3	1	0.9987	3.9502	1.1002	1.0437	1.0054	0.99	0.9956	1.0045	2.7435	1.657	1.745	1.181	1.1592
m = 4	1	0.998	3.6829	1.059	1.0722	0.9984	0.9951	0.9929	0.9924	2.4123	1.616	1.5863	1.087	1.1092
m = 5	1	0.999	3.864	1.0907	1.0944	1.0012	0.9984	1.0161	1.0087	2.5345	1.458	1.4054	1.0529	1.0451
Batch Size														
B = 4	1	0.9987	3.9502	1.1002	1.0944	1.0054	0.9984	0.999	0.9973	2.7435	1.657	1.745	1.181	1.1592
B = 8	1	0.999	3.0562	1.0907	1.0755	0.9944	0.9939	1.0161	1.0087	2.4123	1.543	1.5401	1.0326	1.0337
Jobs/Family														
60	1	0.9978	3.864	1.0412	1.0285	0.9909	0.9984	0.9896	0.9924	2.5345	1.606	1.5401	1.095	1.0928
80	1	0.999	3.6829	1.0907	1.0944	1.0054	0.9982	1.0161	1.0087	2.3591	1.616	1.5863	1.102	1.0967
100	1	0.9987	3.9502	1.1002	1.0722	0.9955	0.9927	0.999	1.0045	2.7435	1.657	1.745	1.181	1.1592
Due-Date Variance														
R = 0.5	1	0.999	3.9502	1.0907	1.0188	0.9944	0.9876	0.9956	0.9886	2.7435	1.492	1.5401	0.9739	0.9686
R = 2.5	1	0.9987	2.2471	1.1002	1.0944	1.0054	0.9984	1.0161	1.0087	2.1826	1.657	1.745	1.181	1.1592
T = 30%	1	0.9987	3.9502	1.0907	1.069	0.9984	0.9939	0.9929	0.9937	2.7435	1.616	1.5863	1.0598	1.0558
T = 60%	1	0.999	2.3079	1.1002	1.0944	1.0054	0.9984	1.0161	1.0087	2.4937	1.657	1.745	1.181	1.1592
Overall	1	0.999	3.9502	1.1002	1.0944	1.0054	0.9984	1.0161	1.0087	2.7435	1.657	1.745	1.181	1.1592

Table 6. Worst case solution ratios.

In both the versions the rules involving EDD (EDD-GA1a-EDD and GA2-EDD-EDD) perform very poorly, giving solutions worse than the ATC-BATC rule. Surprisingly, even the ATC-based algorithms of GA version 2 that do not include swapping (GA2-ATC-BATC and GA2-ATC-DH) perform worse than ATC-BATC. However, when swapping is included, GA2-ATC-DH-Swap and GA2-ATC-BATC-Swap perform reasonably well; on average they produce about 5–6% improvement (see table 5). The ATC-BATC-swap algorithm produces results better than using just ATC-BATC itself, but still does not provide results as good as the those provided by the GA version 1 algorithms (like ATC-GA1a-BATC-swap).

We also analysed the worst-case solution ratios for the algorithms (see table 6). It was found that the algorithms that yield maximum improvement in the average case (ATC-GA1-BATC-swap, ATC-GA1-DH-swap, ATC-GA2-BATC-swap and ATC-GA2-DH-swap) perform at least as good as the ATC-BATC value or just slightly worse than the ATC-BATC in the worst case. The other algorithms have very poor worst-case solution ratios. As expected, the ATC-BATC-swap algorithm which either improves the objective value or leaves it unchanged always has a worst case ratio less than or equal to 1.

5.2. Effect of due date range (R) and number of tardy jobs (T)

As observed before, the values of R and T have a significant impact on the percentage improvements obtained by the GA. The performance of the four best algorithms ATC-GA1a-BATC-swap, ATC-GA1a-DH-swap, ATC-GA1b-BATC-swap, and ATC-GA1b-DH-swap for the four different levels of R and T is shown in table 7.

The improvement (20–25%) at the level R=0.5 and T=30% (see figure 5) was found to be statistically different from the other levels. All of the four algorithms produce 20–25% improvement for this level. Figure 5 shows the level where R=0.5 and T=30%. The bracket in the figure shows the likely spread of the due dates for this level. Notice that the due dates are closely clustered, and are located far from time 0 (start time). The ATC-BATC rule has trouble distinguishing the differences between the slacks of the jobs in this case. Therefore the batching and

Compare	ATC-GA1a-BATC-swap	ATC-GA1a-DH-swap	ATC-GA1b-BATC-swap	ATC-GA1b-DH-swap
R=0.5 & T=30%	0.764	0.748	0.792	0.765
R=0.5 & T=60%	0.959	0.949	0.960	0.946
R=2.5 & T=30%	0.975	0.970	0.974	0.974
R=2.5 & T=60%	0.979	0.951	0.963	0.957

Table 7. Effect of due-date variance and percentage of tardy jobs.

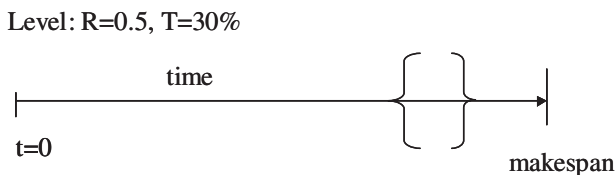


Figure 5. Spread and location of due dates for R=0.5 and T=30%.

sequencing decisions made early on the time axis by the ATC-BATC are often not effective. When more batches are scheduled, and the time gets closer to the location of the due dates, it is easier for ATC to distinguish the slacks, but then the batching and sequencing decisions taken earlier are irrevocable. Algorithms ATC-GA1a-BATC-swap, ATC-GA1a-DH-swap, ATC-GA1b-BATC-swap and ATC-GA1b-DH-swap, which try different batch assignments using the GA and also change batch compositions using swaps, manage to produce a good deal of improvement for this level.

5.3. Computation time

The solution quality of the algorithms was analysed in section 5.1. Algorithms of GA version 1 consistently produced improvement over the ATC-BATC rule. However, since these algorithms use the intensive search procedures of the GA, they consume a good deal of computation time. Table 8 shows the time taken by the CPU of a Pentium III processor (800 MHz, 785 MB) in seconds to run each of the 12 different algorithms for the different factors. The time taken by ATC-BATC and ATC-BATC-swap are also recorded. Both have very low computation times.

Clearly, simple EDD and ATC based rules that do not involve swapping and enumerative procedures like DH take very little computation time. In general, algorithms of GA version 2 type take much longer to generate a solution. The four algorithms that have the best solution qualities are: ATC-GA1a-BATC-swap, ATC-GA1a-DH-swap, ATC-GA1b-BATC-swap, and ATC-GA1b-DH-swap. Of these, the algorithms with DH take about 50% longer than the algorithms with BATC; however their solution qualities are very close. The user can decide whether the relatively small improvements in solution quality justify the additional computational effort.

5.4. GA version 1 versus GA version 2

In GA version 1 the batches are assigned to the machines, while in GA version 2 jobs are assigned to the machines. Clearly in version 1 there are fewer assignment possibilities than in version 2. The superior performance of GA version 1 could be linked to this difference between the two versions. Because it has to explore a large number of possibilities, it is easier for GA version 2 to get stuck in a local optimum. A look at table 8 shows that the computation times of GA version 2 algorithms are much higher than those required for algorithms of GA version 1.

In addition, the performance of a chromosome in GA version 2 is dependent on the assignment of the jobs to the machines while in GA version 1 the batching phase is de-coupled from the assignment phase and precedes it. Therefore, if the batching phase is handled by a good heuristic there remains less work to be done by GA version 1.

For GA version 1 algorithms the improvements provided by Stage 4 (when swaps are carried out) over the first three phases are not as significant as they are for GA version 2 algorithms. In both versions, the batching stage and the sequencing stage use the same heuristics (ATC for batching; BATC and DH for sequencing); the only difference is in the assignment stage. This indicates that in GA version 2 the assignment of jobs to machines is not as effective, and therefore considerable improvement can be achieved in the corrective swap stage.

Compare	ATC BATC	ATC BATC swap	EDD GA1 EDD	ATC GA1a BATC	ATC GA1a DH	ATC GA1a BATC swap	ATC GA1a DH swap	ATC GA1b BATC swap	ATC GA1b DH swap	GA2 EDD EDD	GA2 ATC BATC	GA2 ATC DH	GA2 ATC BATC Swap	GA2 ATC DH swap
Machines														
m = 3	1.08	1.14	1.63	8.75	19.58	15.96	25.08	15.42	26.67	10.83	48.83	65.79	75.29	82.00
m = 4	1.02	1.52	1.75	7.46	17.79	18.67	25.92	17.92	29.33	13.33	40.75	58.21	84.63	91.04
m = 5	0.95	1.56	2.21	7.38	16.71	19.58	28.54	21.88	27.50	14.33	36.08	52.50	86.83	96.92
Batch Size														
B = 4	0.76	1.3	1.47	7.83	18.47	12.83	20.92	12.56	22.86	6.11	33.50	47.17	53.64	65.89
B = 8	0.63	0.75	0.78	2.97	6.44	10.25	13.08	9.67	14.47	10.00	26.22	35.50	52.97	49.47
Jobs/Family														
60	0.6	0.7	0.92	3.42	8.04	8.21	12.13	8.00	12.33	6.08	20.75	30.75	31.17	34.13
80	0.66	1.04	1.25	5.75	12.75	10.54	16.50	11.08	18.67	7.17	30.71	41.79	53.46	58.88
100	0.83	1.33	1.21	7.04	16.58	15.88	22.38	14.25	25.00	10.92	38.13	51.46	75.29	80.04
Due-date variance														
R = 0.5	0.7	1.04	1.25	5.64	12.17	11.14	16.19	10.64	18.58	8.03	29.28	39.89	36.97	41.17
R = 2.5	0.69	1.01	1.00	5.17	12.75	11.94	17.81	11.58	18.75	8.08	30.44	42.78	69.64	74.19
% of Tardy Jobs														
T = 30%	0.66	1.05	0.81	4.86	9.28	8.28	13.39	7.86	13.25	7.22	30.08	39.50	44.33	48.86
T = 60%	0.66	1	1.44	5.94	15.64	14.81	20.61	14.369	24.08	8.89	29.64	43.17	62.28	66.50
Overall	0.7	1.02	1.13	5.40	12.46	11.54	17.00	11.11	18.67	8.06	29.86	41.33	53.31	57.68

Table 8. Computation time for the various algorithms.

6. Conclusions and future research

The main contribution of this paper is the extension of the single-machine batching problem considered by Perez (1999) and Devpura *et al.* (2000) to parallel machines. The inclusion of parallel machines implies that in addition to the batching and sequencing decisions, the decision of assignment (of either jobs or batches to the machines) also needs to be considered. As a solution approach, this paper incorporates the use of a GA hybridized with dispatching rules such as EDD and ATC and scheduling rules such as the decomposition heuristic (DH). Two different versions of the GA (GA version 1 and GA version 2) are proposed based on the decomposition approaches. Additionally, we also extend swapping rules for changing the content of batches suggested by Devpura *et al.* (2000) for a single batch machine, to parallel batch machines. This research establishes the superiority of the algorithms of GA version 1 in both solution quality and computation time through experimentation using a random set of data. Therefore an important conclusion from this paper is that decoupling the phase of batch formation (i.e. forming batches before they are assigned) yields significantly better results, and also saves computation time.

It is also observed that the ATC-BATC rule provides a reasonable solution in a relatively short amount of time even for large-sized problems. Several algorithms in GA version 1 are hybridized with the ATC-BATC rule, and improve upon it in all the cases. The improvements are significant for a particular level of due-date range $R=0.5$ and number of tardy jobs $T=30\%$. For other levels of these factors, improvements were in the range of 5–7%. In semiconductor manufacturing even 1% improvement can cause considerable gains in revenue. Therefore it would be worthwhile to implement these GA-based algorithms in the batch processing areas of a wafer fabrication facility. Depending upon the availability of time, different choices can be made. If a quick solution is required, running the ATC-BATC rule would be reasonable; however, when time is available, better solutions can be obtained using the GA-based algorithms. The spread and location of due dates of jobs is also an important consideration. If the due dates are widely spread we expect the ATC-BATC rule to perform comparably with the GA. However, if the due dates are closely clustered, the GA-based algorithms are a better choice.

Extending the problem to include ready times for jobs is a logical next step for future work. Research is also required on finding good lower bounds so that the performance of the heuristics can be assessed.

The framework suggested for the problem researched in this paper ($P_m | \text{batch, incompatible} | \sum w_j T_j$) can easily be modified to include ready times ($P_m | \text{batch, incompatible, } r_j | \sum w_j T_j$). We can use the same decomposition approaches suggested. However, a heuristic is required to form batches and sequence them in the changed scenario of jobs having ready times. The problem is much more complicated, and several issues need to be addressed. There are two decision points: the time when a machine finishes processing, and the time when a job arrives. At either of these points, a decision has to be made whether to start a batch now or wait for a job or some jobs to arrive. Glassey and Weng (1991) and Fowler *et al.* (1992) illustrate the need to use information of future events for making decisions in a stochastic batch-processing environment. Knowledge of jobs arriving in the future can be obtained from the factory information system. Using 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.

Acknowledgements

This research was supported by the Factory Operations Research Center which is jointly funded by International SEMATECH and the Semiconductor Research Corporation (SRC) under contract 2001-NJ-880. Parts of this research were carried out when the second author was visiting the Modeling and Analysis of Semiconductor Manufacturing (MASM) Laboratory at Arizona State University, Tempe.

References

- AZIZOGLU, M. and WEBSTER S., 2001, Scheduling a batch processing machine with incompatible job families. *Computers & Industrial Engineering*, **39**, 325–335.
- DEV PURA, A., FOWLER, J. W., CARLYLE, M. and PEREZ, I., 2000, Minimizing total weighted tardiness on a single batch processing machine with incompatible job families. *Proceedings of the Symposium on Operations Research*, Dresden, Germany, pp. 366–371.
- DUPONT, L. and DHAENENS-FLIPO, C., 2002, Minimizing the makespan on a batch machine with non-identical job sizes: an exact procedure. *Computers & Operations Research*, **29**, 807–819.
- FOWLER, J. W., PHILLIPS, D. T. and HOGG, G. L. R., 1992, Real-time control of multiproduct bulk-service semiconductor manufacturing processes. *IEEE Transactions on Semiconductor Manufacturing*, **5**, 158–163.
- FOWLER, J. W., HORNG, S. and COCHRAN, J. K., 2001, A genetic algorithm approach to manage ion implantation processes in wafer fabrication. *International Journal of Manufacturing Technology Management*, **1**, 156–172.
- GLASSEY, C. R. and WENG, W. W., 1991, Dynamic batching heuristic for simultaneous processing. *IEEE Transactions on Semiconductor Manufacturing*, **4**, 77–82.
- GRAVEL, M., PRICE, W. L. and GAGNE, C., 2000, Scheduling jobs in an Alcan aluminum foundry using a genetic algorithm. *International Journal of Production Research*, **38**, 3031–3041.
- LAWLER, E. L., 1977, A ‘Pseudopolynomial’ time algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, **2**, 75–90.
- MEHTA, S. V. and UZSOY, R., 1998, Minimizing total tardiness on a batch processing machine with incompatible job families. *IIE Transactions*, **30**, 165–178.
- MICHALEWICZ, Z., 1996, *Genetic Algorithms + Data Structures = Evolution Algorithms*, 3rd edn (Berlin, Heidelberg and New York: Springer).
- MÖNCH, L., 2002, A genetic algorithm heuristic applied to stepper scheduling. *Proceedings of the International Conference on Modeling and Analysis of Semiconductor Manufacturing (MASM)*, Tempe, Arizona, pp. 276–281.
- MYERS, R. H., MONTGOMERY, D. C., 2002, *Response Surface Methodology* (New York: Wiley).
- PINEDO, M., 1995, *Scheduling: Theory, Algorithms and Systems* (Englewood Cliffs, NJ: Prentice Hall).
- PEREZ, T., 1999, Minimizing total weighted tardiness on a single batch process machine with incompatible job families. Master’s thesis, Arizona State University, Tempe, Arizona.
- VEPSALAINEN, A. P. J. and MORTON, T. E., 1987, Priority rules for job shops with weighted tardiness costs. *Management Science*, **33**, 1035–1047.
- WALL, M., 1999., *Galib: A C++ library of Genetic Algorithms Components*, Website: <http://lancet.mit.edu/ga/>.
- WANG, C. and UZSOY, R., 2002, A genetic algorithm to minimize maximum lateness on a batch processing machine. *Computers & Operations Research*, **29**, 1621–1640.