Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)

Multi-Objective Semiconductor Manufacturing Scheduling: A Random Keys Implementation of NSGA-II

Scott J. Mason, Mary E. Kurz, Michele E. Pfund, John W. Fowler, *Member, IEEE*, Letitia M. Pohl

***Abstract*—We examine a complex, multi-objective semiconductor manufacturing scheduling problem involving two batch processing steps linked by a timer constraint. This constraint requires that any job completing the first processing step must be started on the succeeding second machine within some allowable time window; otherwise, the job must repeat its processing on the first step. We present a random keys implementation of NSGA-II for our problem of interest and investigate the efficacy of different batching policies in terms of the number of approximate efficient solutions that are produced by NSGA-II over a wide range of experimental problem instances. Experimental results suggest a full batch policy can produce superior solutions as compared to greedy batching policies under the experimental conditions examined.**

I. Introduction

THE problem addressed in this paper is motivated by an application in semiconductor manufacturing. We focus on a flexible flow shop with four machines, as pictured in Figure 1. Machine A is capable of processing up to six jobs simultaneously as a batch. It feeds the three parallel downstream machines: B, C and D, each of which has a batching capacity of its own. Between machine A and the downstream machines, there is a timer. A job that completes processing on machine A must begin processing on the next machine in its route before a fixed time period. If the queue time at the downstream machine exceeds that time limit, the job must be processed again on machine A. This is referred to as recirculation. The goal is to prevent this recirculation, if possible, to avoid additional congestion in the system.

There are three types of jobs: B, C and D. All jobs are processed on machine A. But due to machine eligibility restrictions, jobs must be processed on a particular downstream machine. Type B jobs are processed by machine A, and then processed by machine B. Likewise, type C and D jobs are processed by machine A and then by machines C and D, respectively. Machine eligibility restrictions are denoted by M_j, where M_j is the set of machines that can process job j. Additionally, each job has two batch identification (ID) codes associated with it. The upstream batch ID identifies the processing time on machine A, while the downstream batch ID identifies the eligible downstream machine (B, C or D) and the downstream processing time. The upstream machine uses a compatible batching concept. In compatible batching, jobs of different types and batch IDs can be processed together, with the batch processing time being equal to the longest individual processing time of the jobs in the batch. The downstream machines use an incompatible batching concept. Not only do batched jobs have to be of the same type, but they also must share the same ID code. Jobs that have different ID codes are considered incompatible and cannot be batched together.

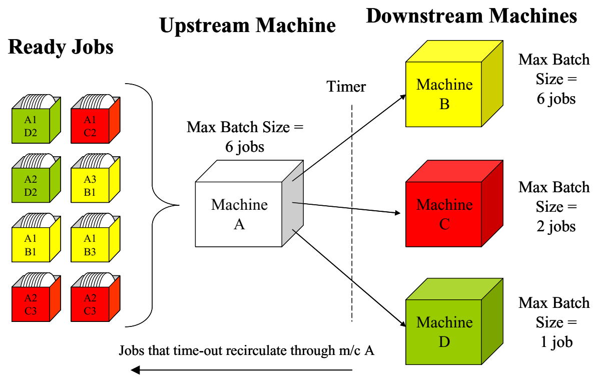

Fig. 1. Job Routing With Batching in a Flexible Flow Shop

The goal of our analysis is to develop heuristic approaches to improve on-time delivery, reduce cycle time variation, and reduce violation of the timer to minimize recirculation. The achievement of these goals is dependent on how effectively the jobs are batched at both the upstream and downstream machines. Clearly, the timer plays an important part in determining how batches are formed, as the downstream queue length and/or downstream processing times can cause timer violations.

Each job, denoted by j, has several parameters associated with it. It has priority or weight w_j, ready time r_j, due date d_j, a batch ID code for the upstream machine and a batch ID code for the downstream machine. The upstream batch ID determines the upstream job processing time p_{1j}. Likewise, the downstream batch ID determines the downstream job processing time p_{2j}.

In practically motivated problems it is typical to have

Manuscript received October 31, 2006.
Scott J. Mason and Letitia M. Pohl are with the University of Arkansas, Fayetteville, AR 72701 USA (479-575-5521; e-mail: mason@uark.edu and lpohl@uark.edu)
Mary E. Kurz is with Clemson University, Clemson, SC 29634 USA (e-mail: mkurz@clemson.edu)
Michele E. Pfund and John W. Fowler are with Arizona State University, Tempe, AZ 85287 USA (e-mail: michele.pfund@asu.edu and john.fowler@asu.edu)

multiple objectives. In this scenario, we seek to minimize three objectives: 1) Total Weighted Tardiness ($\sum_j w_j T_j$), where job *j*'s tardiness $T_j = \max(C_j - d_j, 0)$, with C_j denoting job *j*'s completion time; 2) Cycle Time Variation, where cycle time $F_j = C_j - r_j$ and $Var(F_j) = \frac{1}{n-1}\left(\sum_j F_j^2 - n\bar{F}^2\right)$; and 3) Cumulative Timer Violation $\sum_j V_j = \sum_j \max\left[\left(C_j - p_{2j}\right) - \left(s_j + p_{1j}\right) - \tau, 0\right]$, where the job start time on the downstream machine is $C_j - p_{2j}$, the job completion time on the upstream machine is $s_j + p_{1j}$ and the timer length is τ. Generally, one solution will not simultaneously minimize our objectives. The concept of efficient solutions is used to identify solutions that may be desirable to a decision maker. A solution σ is weakly efficient (with respect to the criteria of interest, here called z_1 and z_2, which are to be minimized) if there is no other solution σ' such that $z_1(\sigma) \le z_1(\sigma')$ and $z_2(\sigma) \le z_2(\sigma')$. A solution σ is efficient if it is weakly efficient and at least one of the previous relations holds as a strict inequality. We then say that solution σ' dominates solution σ. These definitions are easily extendable to the multi-criteria case.

The use of GAs in finding the set of efficient solutions seems especially desirable because GAs evolve sets of solutions and multi-criteria problem solvers may want the entire set of efficient solutions. Since it is not guaranteed that any heuristic will find optimal solutions to NP-hard single criterion optimization problems, GAs can only find approximately efficient solutions (AES) to multi-criteria optimization problems. Using the scheduling problem classification scheme of Lawler *et al.* [1], our problem is $FF2 \mid r_j, M_j, p-batch, compatible, incompatible \mid \sum w_j T_j, Var(F_j), \sum V_j$. Pinedo [2] shows through reduction of the 3-PARTITION problem that $1 \| \sum w_j T_j$ is strongly NP-hard. Since our problem can be thought of as a special case of the single machine problem, $1 \| \sum w_j T_j$ reduces to our problem and we can conclude that its complexity is at least strongly NP-hard. Therefore, in this paper, we implement NSGA-II [3] using a random keys chromosome representation as an initial step in our analysis, placing our primary focus on the generation of approximately efficient solutions to this multi-objective scheduling problem.

II. Previous Research

A. Batching Heuristics

Previous research has concentrated on the batch machine dispatching problem. The minimum batch size (MBS) rule was proposed by Neuts [4]. MBS rules specify that an incomplete load can be processed once the queue size reaches a specified minimum. Techniques have been proposed to optimize that minimum value. The MBS rule is considered a theoretical standard used to evaluate the performance of other batch dispatching rules [5]. Batching heuristics seek to minimize wasted capacity resulting from a load that is smaller than the machine capacity, while minimizing machine idle time that may be required to form a batch.

Glassey and Weng [6] present the dynamic batching heuristic (DBH) to minimize the average delay at the batching machine. The next arrival strategic control heuristic (NACH) was developed by Fowler *et al.* [7]. The NACH expands on DBH by implementing a rolling horizon policy, where only the next arrival is considered. This makes it less susceptible than DBH to arrival prediction error. A similar heuristic, the minimum cost rate (MCR) heuristic, was proposed by Weng and Leachman [8]. The MCR heuristic is based on minimizing a holding cost per unit time. Finally, the rolling horizon cost rate (RHCR) heuristic [9] combines the rolling horizon used in NACH with the cost rate function used in MCR.

Something that all these batching heuristics (except MBS) have in common is that they consider upstream information, or knowledge of future arrivals. Extensions of two of these heuristics consider downstream information. NACH-setup is an extension of the NACH policy and takes into account downstream setup times [5]. Later, decision logic was added to RHCR that uses information about the state of the serial machine, in an attempt to further reduce flowtimes [9]. The objective function in this case was average standardized flowtime, where the time from job arrival to completion on the serial machine is divided by the sum of its batch and serial processing times.

B. Dispatching Rules and Look-Ahead Dispatching Rules

Dispatching rules are used to find reasonably good solutions is a short period of time, and can therefore be used for real-time control of job flow. They provide a method to select the job to be processed next from a set of waiting jobs. Some traditional dispatching rules typically considered include earliest due date (EDD, non-decreasing order of d_j), weighted shortest processing time (WSPT, non-increasing order of w_j/p_j), and minimum slack (MS, non-decreasing order of $\max(d_j - p_j - t, 0)$ for each job).

These rules are known as *local* dispatching rules because they only require information about the jobs currently waiting at the machine in question. *Global*, or *look-ahead* dispatching rules require information about jobs at other machines. These look-ahead rules look forward in time and can consider information either upstream or downstream. A simple look-ahead rule that takes advantage of downstream information is shortest queue at the next operation. With this rule, every time a machine is freed, the job with the shortest queue at the next machine on its route is selected for processing [2].

Some look-ahead procedures look one, or even a few steps ahead in time and evaluate the impact of the current decision on the objective function. This look ahead in time can be to access upstream information, downstream information, or both. The batching rules discussed previously are a specific type of look-ahead rule. Jang [10] reviews a number of existing look-ahead procedures that were developed for problem specific scenarios. Finally, Holthaus and Ziegler [11] develop a coordinating dispatching rule called look ahead job demanding (LAJD). They define a coordination rule as one that requires coordination of machines, and therefore considers not only the jobs waiting in queue for the current machine, but also the states of all the machines preceding the current machine.

III. Genetic Algorithms

A. NSGA-II

The complexity of the problem precludes the possibility of building an optimization model without oversimplifying the problem. For this reason, we implement NSGA-II using a random keys chromosome representation. NSGA-II was published by Deb *et al.* [3] and is considered by many to be the dominant evolutionary multi-objective optimization algorithm. A complete description of NSGA-II is in Deb *et al.* [3], but we summarize the key points for the convenience of the reader. We also include an adaptation of Figure 2 from Deb *et al.* [3] as Figure 2 below.

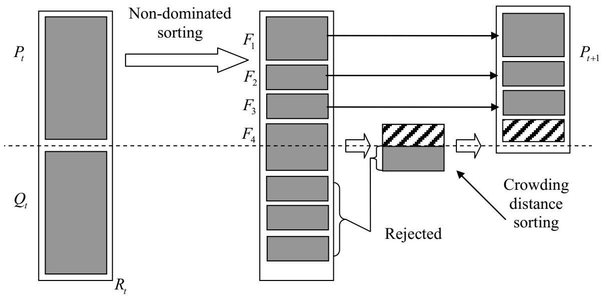

Fig. 2. Adaptation of Figure 2 from Deb *et al.* [3]

NSGA-II utilizes the current set of N solutions in generation t, P_t, to build an offspring set of N solutions Q_t, which are combined to create a larger set of $2N$ solutions called R_t. The method by which we build Q_t differs from that used by Deb *et al.* [3], so we delay that discussion until a later section. Once R_t is built, the N "best" solutions are kept to create the next set of solutions, P_{t+1}. Two steps are used to find the "best" solutions: non-dominated sorting and crowding distance sorting.

Non-dominated sorting can be explained as follows. Consider two objectives, z_1 and z_2 and a set of $2N$ solutions, R_t, generated in some manner. The solutions in R_t which are not dominated by any other solutions are said to be in front 1, F_1. The solutions in R_t minus F_1 which are not dominated by any other solutions in R_t minus F_1 are said to be in front 2, F_2. There can be at most $2N$ fronts. Deb *et al.* [3] give an $O(M(2N)^2)$ procedure for identifying the fronts, where M is the number of objectives under consideration. In practice we need only identify fronts until N solutions have been included; if fronts 1 through f-1 have fewer than N members and fronts 1 through f have N or more members, we will not retain the remaining members of R_t in P_{t+1}. The members of F_f which are retained in P_{t+1} are selected using crowding distance sorting.

Crowding distance sorting intends to keep a diverse set of solutions from F_f. Figure 3 shows five solutions in an arbitrary front with two objectives. Consider solution 3. The crowding distance for solution 3 is $3_{\text{distance}} = \dfrac{z_1(4)-z_1(2)}{z_1(5)-z_1(1)} + \dfrac{z_2(2)-z_2(4)}{z_2(1)-z_2(5)}$. The crowding distance for solutions 1 and 5 are ∞. In Figure 3, we see that solution 2 is in a more crowded region of the front, and solution 4 is in a less crowded region of the front. Solution 4's distance metric will be larger than solution 2's, so we will retain solution 4 in P_{t+1} before retaining solution 2 in P_{t+1}.

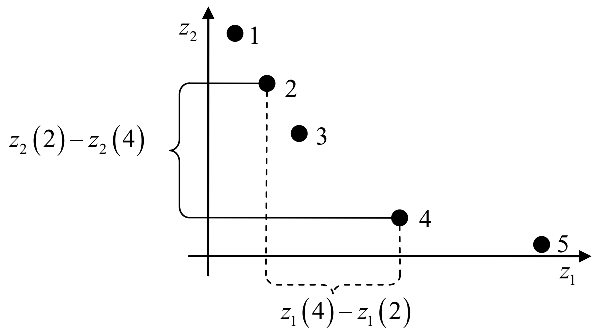

Fig. 3. Crowding Distance Sorting Example

We describe the method for an arbitrary front F_i of l

solutions, following Deb *et al.* [3] closely.

```
For each solution i, $i_{\text{distance}} = 0$
For each objective m
Sort the solutions in increasing order.
Set the crowding distance of the first and last sorted solutions: $[1]_{\text{distance}} = [l]_{\text{distance}} = \infty$
Set the crowding distance of the rest of the solutions, i=2 to l-1: $[i]_{\text{distance}} = [i]_{\text{distance}} + \frac{z_m(i+1) - z_m(i-1)}{z_m(l) - z_m(1)}$
```

B. A Random Keys Implementation of NSGA-II

The random keys representation was introduced by Bean [12] and has been used in numerous applications of combinatorial optimization ([13], [14], [15]). While the traditional (binary) GA encoding is well-suited to problems with real-valued decision variables, combinatorial optimization problems often have solutions that can be represented by a permutations. Unfortunately, a direct permutation encoding combined with traditional genetic operators can easily lead to chromosomes representing infeasible solutions, which must then be repaired. The random keys encoding avoids the problem of infeasible chromosomes. In our application, each chromosome consists of one gene per job. The values of the genes are generated from a uniform distribution [0, 1000]. These genes serve as *sort keys* used to determine the order in which jobs are processed on machine A. The jobs are ordered for machine A in increasing order of the sort keys. Batches are formed for machine A using this order and one of two batching options:

1) Full batching – each batch will contain the maximum number of jobs allowed by the batch capacity, in this case maximum batch size is six.

2) Minimum (greedy) batching – the batch will contain as many jobs as are ready at the time the batch begins processing (i.e., when the batching machine completes its current batch and becomes available to process a new batch), up to the maximum batch size.

Finally, downstream batches are formed using greedy batching (minimum). These two batching options are investigated due to their prevalent use in the semiconductor industry. All parameter settings were chosen empirically to balance running time and solution quality. The following text describes our random keys implementation of NSGA-II in detail.

Initialization—The initial population P_0 of size *N* is generated randomly.

Creation of Q_t—In contrast to Deb *et al.* [3], we use the same mechanism to create the offspring population Q_t for all values of *t* and we utilize parametric uniform crossover and immigration (following Bean [12]) to create Q_t from P_t. We use parametric uniform crossover to create 60% of the members of Q_t and use immigration (randomly generating new chromosomes to create 40% of the members of Q_t. Parametric uniform crossover selects two different parents randomly from P_t to create one new member of Q_t. Each gene is selected from parent 1 (designated randomly) with a 70% probability, otherwise it is selected from parent 2.

Creation of P_{t+1}—Once the chromosomes are decoded, the total weighted tardiness, cycle time variation and cumulative timer violations are computed. We faithfully implement non-dominated sorting and crowding distance sorting, but we insert a *duplicate removal* procedure between those steps. Consider Figure 4, in which two solutions (3 and 4) have objective functions that map to the exact same place in the solution space. In combinatorial problems with integer data, this is anecdotally common. We assume that the solutions shown are in front F_f and that only 4 solutions will be retained to form P_{t+1}. Table 1 contains data for this example.

If we retain exactly four solutions from this front in P_{t+1}, the solutions retained will differ if solution 3 is included for consideration or not. In the first case, when all 6 solutions are considered, solutions 1, 6, 5 and 2 or 4 will be retained. Solutions 2 and 4 have the exact same crowding distance ($4/14 + 5/12$) but correspond to different points in the solution space. In the second case, when solution 3 is not considered, we see that solutions 1, 6, 5 and 4 will be retained; solutions 2 and 4 may now be differentiated in terms of their relative crowding in the front. It is our contention that not including all duplicates in the solution space when computing crowding distances better reflects the intent of the crowding distance metric for diversity preservation. Therefore, we will strip duplicates in the solution space from fronts created from R_t before computing the crowding distances.

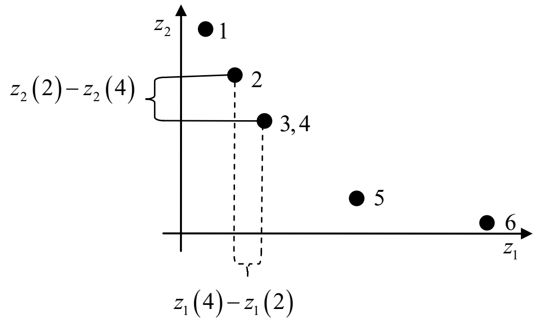

Fig. 4. Creation of P_{t+1} Example

TABLE 1. DATA FOR EXAMPLE PROBLEM IN FIGURE 4

Solution i	z_1	z_2	i_{distance} (3 and 4 considered)	i_{distance} (3 not considered)
1	0	13	∞	∞
2	3	10	0.702	0.702
3	4	8	0.238	--------
4	4	8	0.702	0.940
5	8	3	1.298	1.298
6	14	1	∞	∞

Stopping Criteria—We stop if 400*N* or more chromosomes have been evaluated. If fewer than 400*N* chromosomes have been evaluated, we go to create the next offspring population Q_t.

IV. EXPERIMENTAL DESIGN

In order to evaluate the performance of our random keys implementation of NSGA-II, a series of computational experiments were run using randomly generated test problems. The experimental factors and their associated factor levels at both Low and High settings are defined as follows:

Number of Jobs (n): Low = 60, High = 120

Ready Times (r_j): Low = 0, High = 50% 0, 50% DU[1, $C_{\max}(est)$], where

$$C_{\max}(est) = \max\left\{\begin{matrix} \frac{n}{b_A}E[p_{Aj}] + \max_{i\in\{B,C,D\},j} p_{ij}, \\ E[p_{Aj}] + \max_{i\in\{B,C,D\},j}\left(\frac{n}{b_i}E[p_{ij}]\right)\end{matrix}\right\}$$

Due Date (d_j): DU$\left[\mu - \frac{\mu R}{6}, \mu + \frac{\mu R}{6}\right]$, where $\mu = C_{\max}(est)$(1-*T*), *T*: Low = 0.45, High = 0.9, and *R*: Low = 0.5, High = 2.5

In all experiments, job weight w_j is distributed according to a discrete uniform distribution over the integer set [1,10], population size $N = 100$, and timer length $\tau = 2$. Machines A, B, C, and D have maximum batch sizes of 6, 6, 2, and 1 jobs, respectively. Further, batch ID values are randomly assigned so that a job has an equal probability of 1) having any one of machine A's three batch IDs and 2) being assigned to machine B, C, or D for its second step. Also, a job is equally likely to be assigned any batch ID corresponding to the tool to which it is assigned. The process times p_j associated with each machine's batch IDs are as follows:

Machine A: A1 = 2 hours, A2 = 2.5 hours, A3 = 3 hours

Machine B: B1 = 6 hours, B2 = 8 hours, B3 = 10 hours

Machine C: C1 = 1 hour, C2 = 2 hours, C3 = 3 hours

Machine D: D1 = 0.5 hours, D2 = 0.75 hours, D3 = 1 hour

We generate 10 problem instances for each factor combination, resulting in a total of 2(2)2(2)10=160 test cases. Each test case is run through our NSGA-II implementation a total of 50 times (i.e., 50 different initial population replications) using both Greedy and Full batching policies on machine A.

V. ANALYSIS FOR MULTIPLE OBJECTIVES

Consider a test case of our problem of interest. The primary performance measure reported by our NSGA-II implementation in the number of approximately efficient solutions (AES). Let $AES(\lambda, \pi, P)$ denote the number of AESs found in replication π ($\pi = 1...50$) of problem instance λ ($\lambda = 1...10$) using batching policy *P* ($P \in \{Greedy, Full\}$) for a given test case. Table 2 reports the experimental results in terms of $\overline{AES} = \frac{1}{10}\sum_{\lambda=1}^{10}\left(\frac{\sum_{\pi=1}^{50} AES(\lambda,\pi,P)}{50}\right)$ for a specific experimental factor of interest. Further, let $SD(\overline{AES})$ denote the standard deviation of the $\overline{AES}$ values for a specific experimental factor of interest. For example, the first row of data in Table 2 depicts the $\overline{AES}$ and $SD(\overline{AES})$ values for each batching policy *P* when the number of jobs factor is at its low level (i.e., 60 jobs)—all other factors are aggregated in this initial data row.

From Table 2, we note that while the 95% confidence intervals indeed overlap in all cases for the two machine A batching policies under consideration, the mean values for the full batching policy are consistently lower than those for greedy batching. Further, in the majority of our experimental cases, the full batching policy also produced a more consistent number of AESs, as its $SD(\overline{AES})$ values are lower than those for Greedy batching policy.

In terms of required computation time, our NSGA-II implementation required approximately 35 seconds to complete 50 replications of each 60-job test case and 65 seconds to complete a 120-job test case replication.

VI. CONCLUSIONS

The problem addressed in this paper is motivated by an application in semiconductor manufacturing. We examine a complex, multi-objective semiconductor manufacturing scheduling problem involving two batch processing steps linked by a timer constraint. This constraint requires that any job completing the first processing step must be started on the succeeding second machine within some allowable time window; otherwise, the job must repeat its processing on the first step.

TABLE 2. COMPUTATION RESULTS

		Machine A Batch Policy			
		Greedy		Full	
Factor	Level	$\overline{AES}$	$SD(\overline{AES})$	$\overline{AES}$	$SD(\overline{AES})$
Number of Jobs	Low	77.51	12.74	73.11	11.15
	High	84.47	12.61	81.40	10.66
Ready Times	Low	73.43	14.02	73.74	14.03
	High	88.55	5.79	80.77	7.14
T	Low	77.68	16.17	73.40	13.88
	High	84.30	7.88	81.11	7.06
R	Low	78.04	16.51	74.03	14.53
	High	83.94	7.47	80.48	6.37

The complexity of the problem precludes the possibility of building an optimization model without oversimplifying the problem. For this reason, we implement NSGA-II [3] using a random keys chromosome representation. We investigate two different batching policies on the upstream machine: 1) Full batching (each batch contains the maximum number of jobs allowed by the batch capacity) and 2) Minimum or greedy batching (the batch will contain as many jobs as are ready at the time the batch begins processing, up to the maximum batch size).

In this initial step in our experimentation, we are interested in assessing the number of approximate efficient solutions that are produced by NSGA-II over a wide range of experimental problem instances for each batching policy. Experimental results suggest a full batch policy can produce superior solutions as compared to greedy batching policies under the experimental conditions examined. However, this difference is not statistically significant, as the corresponding 95% confidence intervals about the mean values overlap.

Future work is in progress to examine other machine A batching policies, including an adaptation of NACH for compatible jobs. In addition, a detailed heuristic analysis is warranted to assess the true quality of our NSGA-II results in terms of existing, potentially inferior but definitely more computationally attractive dispatching rules commonly used in semiconductor manufacturing practice.

REFERENCES

[1] Lawler, E.L., Lenstra, J.K. and Rinnooy Kan, A.H.G., Recent developments in deterministic sequencing and scheduling: a survey. In Deterministic and Stochastic Scheduling, edited by M.A.H. Dempster, J.K. Lenstra and A.H.G. Rinnooy Kan, (1982), 35–73 (Reidel, Dordrecht).

[2] Pinedo, (1995) *Scheduling: Theory, Algorithms and Systems*, Prentice Hall, NJ.

[3] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan, A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II, *IEEE Transactions On Evolutionary Computation*, 6 (2) (2002) 182-197.

[4] M.F. Neuts, A general class of bulk queues with Poisson input. *Annals of Mathematical Statistics*, 38 (1967) 759-770.

[5] L. Solomon, J. Fowler, M. Pfund and P. Jensen, The inclusion of future arrivals and downstream setups into wafer fabrication batch processing decisions, *Journal of Electronics Manufacturing*, 11 (2) (2002), 149-159.

[6] C.R. Glassey and W.W. Weng, Dynamic batching heuristic for simultaneous processing, *IEEE Transactions on Semiconductor Manufacturing*, 14 (2) (1991), 77-82.

[7] J.W. Fowler, D.T. Phillips and G.L. Hogg, Real time control of multiproduct bulk service semiconductor manufacturing processes, *IEEE Transactions on Semiconductor Manufacturing*, 5 (2) (1992),158-163.

[8] W.W. Weng and R.C. Leachman, An improved methodology for real-time production decision at batch-process work stations, *IEEE Transactions on Semiconductor Manufacturing*, 6 (1993), 219-225.

[9] J.K. Robinson, J.W. Fowler and J.F. Bard, The use of upstream and downstream information in scheduling semiconductor batch operations, *International Journal of Production Research.*, 33 (7) (1995), 1849-1869.

[10] J. Jang, J. Suh, M. Park, and R. Liu, A Look-Ahead Routing Procedure for Machine Selection in a Highly Informative Manufacturing System, *The International Journal of Flexible Manufacturing Systems*, 13 (2001), 287-308.

[11] O. Holthaus and H. Ziegler, Improving job shop performance by coordinating dispatching rules, *International Journal of Production Research*, 35 (2) (1997), 539-549.

[12] Bean, J.C., Genetic algorithms and random keys for sequencing and optimization, *ORSA Journal on Computing*, 6 (1994), 154-160.

[13] Norman, B.A. and Bean, J.C., A genetic algorithm methodology for complex scheduling problems, *Naval Research Logistics*, 46 (1999), 199-211.

[14] Wang, C-S. and Uzsoy, R., A genetic algorithm to minimize maximum lateness on a batch processing machine, *Computers & Operations Research*, 29 (2002), 1621-1640.

[15] Kurz, M.E. and Askin, R.G., Scheduling flexible flow lines with sequence-dependent setup times, *European Journal of Operational Research*, 159 (1) (2004), 66-82.