J Intell Manuf (2012) 23:2271–2280
DOI 10.1007/s10845-011-0571-z

Scheduling two-stage hybrid flow shops with parallel batch, release time, and machine eligibility constraints

I.-Lin Wang · Taho Yang · Yu-Bang Chang

Received: 1 August 2010 / Accepted: 20 June 2011 / Published online: 22 July 2011

Abstract This paper investigates a difficult scheduling problem on a specialized two-stage hybrid flow shop with multiple processors that appears in semiconductor manufacturing industry, where the first and second stages process serial jobs and parallel batches, respectively. The objective is to seek job-machine, job-batch, and batch-machine assignments such that makespan is minimized, while considering parallel batch, release time, and machine eligibility constraints. We first propose a mixed integer programming (MIP) formulation for this problem, then gives a heuristic approach for solving larger problems. In order to handle real world large-scale scheduling problems, we propose an efficient dispatching rule called BFIFO that assigns jobs or batches to machines based on first-in-first-out principle, and then give several reoptimization techniques using MIP and local search heuristics involving interchange, translocation and transposition among assigned jobs. Computational experiments indicate our proposed re-optimization techniques are efficient. In particular, our approaches can produce good solutions for scheduling up to 160 jobs on 40 machines at both stages within 10 min.

Keywords Flow shop with multiple processors · Scheduling · Dispatching · Mixed integer programming · Heuristic algorithm · Makespan

I.-L. Wang (✉)
Department of Industrial and Information Management, National Cheng Kung University, No1. University Rd, Tainan 701, Taiwan
e-mail: ilinwang@mail.ncku.edu.tw

T. Yang · Y.-B. Chang
Institute of Manufacturing Information and Systems, National Cheng Kung University, No1. University Rd, Tainan 701, Taiwan

Introductions

Over the decades, semiconductor manufacturing has become the most capital and labor intensive industry in Taiwan. Besides investigating new manufacturing technologies to boost the productivity, the semiconductor manufacturers also make much effort in promoting their quality of services by improving the production quality and shortening the production time to maintain their competitive edge. Since the equipments for processing wafer is extremely expensive, how to maximize the productivity and utilization of equipments within shorter time become critical issues in semiconductor manufacturing industries. To this end, different techniques such as seeking the optimal lot sizes, Work-In-Process levels, shop floor control, and production scheduling have been widely investigated.

The process of wafer manufacturing is a FSMP (Flow shop with multiple processors) problem, in which the products must go through specific procedures, and pass through the processors in specific orders. Note that jobs could follow very different routing in the production systems due to the re-entrant characteristic of semiconductor manufacturing, which makes the flow-shop assumption restrictive in semiconductor manufacturing. Different products may go through the processors in the same order but are processed by different recipes. To speed up the manufacturing process, machines that can process batch jobs are widely used. In particular, a batch process may handle two or more jobs of the same recipes at the same time, but jobs of different recipes can not be put into the same batch. The batch job process has become a common practice in semiconductor manufacturing industry. Note that we assume zero setup time for changing recipes, since in some manufacturing process different recipe simply refers to different length of time for some chemical processing. Take the wet etching process for example; jobs

of different recipes go through the same etching process but with different lengths of processing time.

Although the introduction of batch process aims to save more processing time, a poor job-machine dispatching rule may force some early jobs to wait for too long and incur longer machine idle time. Such a job-batch and batch-machine scheduling problem becomes even more complicated, when we take the job-machine scheduling decisions of previous stage (e.g. the photoresist stripping process) into consideration.

The scheduling for FSMP is a common and complex problem in semiconductor and electronic manufacturing industry. This paper investigates a specialized two-stage hybrid FSMP problem of M machines in each stage that process N jobs, each with a given release time. This problem appears very often in semiconductor manufacturing industry. In practice, greedy dispatching rules are commonly used to generate a quick scheduling without any performance guarantee. Here we first give a mixed integer programming (MIP) formulation for this problem to calculate a theoretical optimal scheduling. Unfortunately, our MIP formulation can only deal with problems of very small scales (e.g. $M = 2,\ N = 8$) within a tolerant time interval (e.g. 10 min). Such a poor performance is surely unacceptable, especially in semiconductor manufacturing whose daily production planning is often dynamically altered.

To the best of our knowledge, none of previous literatures considers parallel batch, release time, and machine eligibility at the same time, to deal with FSMP. Li and Lee (1997) schedule the semiconductor bum-in operations on ovens that can process batch jobs. Assuming the job release times and due dates are agreeable, they show the problem is strongly NP-hard for specific objective functions. Potts and Kovalyov (2000) give comprehensive review on the literature of scheduling with batching and propose efficient dynamic programming algorithms for solving these types of problems. Mathirajan and Sivakumar (2006) review papers that schedule batch processors in semiconductor manufacturing and propose schemes to classify scheduling problems. They also carry out a simple meta-analysis to understand the development and evolution of research in this topic, which helps to identify potential research areas.

Dobson and Nambimadom (2001) proposes an MIP model for $1 \mid B_j \mid C_{\max}$ scheduling problems, and shows its complexity to be NP-hard; Hung (1998) proposes a dynamic programming approach to the $\mathrm{Pm} \mid B_j \mid L_{\max}$ scheduling problem. Among the literatures using dispatching rules and heuristic approaches, Centeno and Armacost (2004) compare the longest processing time first rule (LPT), the least flexible job first rule (LFJ), the least flexible machine first rule (LFM) and their combination, finding that LPT has the best results. Malve and Uzsoy (2007) propose a genetic algorithm (GA) for $\mathrm{Pm} \mid B_j \mid C_{\max}$ problems. Kashan et al. (2008) solve $\mathrm{Pm} \mid B_j \mid C_{\max}$ problems by integrating GA with some heuristic approach. Bellanger and Oulamara (2009) propose three heuristic approaches to solve p-batch(II)$\mid C_{\max}$ problems. In this paper, we focus on the scheduling problem of FH2B(m1,m2)|p-batch(II), $M_j, r_j \mid C_{\max}$.

As for literatures that solve FSMP by dispatching rules, Hunsucker and Shah (1994) simulate and construct an FSMP environment, and compare six dispatching rules. Uzsoy et al. (1992) test and compare eight dispatching rules for scheduling problems in semiconductor manufacturing. Grangeon et al. (1999) construct an objected oriented FSMP simulation model, and compare the performance of four dispatching rules in thirteen scenarios. Petroni and Rizzi (2002) evaluate the performance of five dispatching rules for solving an FSMP based on fuzzy theories Kuo et al. (2007) simulate Metamodel by neural network to simplify the complexity of the simulation model.

In order to deal with the quality disadvantages of traditional dispatching rules and overcome the efficiency difficulties of MIP formulation, we propose several heuristics. In particular, we first give a technique to reduce the size of the original MIP formulation for the special case where all the jobs have zero release time, so that the reduced formulation, named MIPH, can solve larger scheduling problems within the tolerant time interval. To deal with much larger scheduling problems, we give an efficient dispatching rule called BFIFO that assigns jobs or batches to machines based on First-In-First-Out (FIFO) principle. To further improve the solution quality of BFIFO, we propose and test two mechanisms by: (1) integrating BFIFO and MIPH, and (2) integrating BFIFO and reoptimization heuristics based on local search techniques that involve translocation, interchange and transposition (TIT) among assigned jobs.

The rest of the paper is organized as follows: sect. "Problem definition and mathematical modeling" illustrates our MIP model as well as a heuristic to obtain a reduced MIP model called MIPH. Section "Heuristics based on dispatching rules and local search techniques" describes our BFIFO dispatching rule, and explain how the integration of BFIFO to MIPH and the TIT local search heuristics work; the results of computational experiments on different solution methods are listed in sect. "Computational experiments and analyses". Section "Conclusions" concludes this paper and suggests topics for future research.

Problem definition and mathematical modeling

Problem definition

Most of the semiconductor manufacturing processes are conducted either in serial jobs or in batch jobs. If a job contains 25 wafers, a serial process will manufacture the wafers one

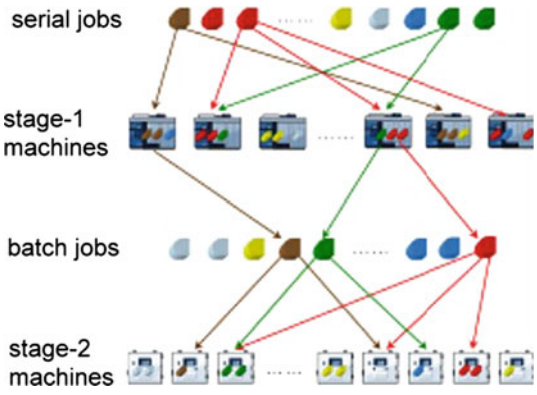

Fig. 1 An illustration of our two-stage hybrid flow shops

by one till all of them are done, and then proceed to the next machine. On the other hand, a batch process can manufacture two to six jobs of the same recipe at the same time. (In this paper, the machines at the second stage can manufacture two jobs simultaneously.) Manufacturing by batches does shorten the completion time of jobs. However, when all the jobs have different release time, whether to save some processing time by waiting for another job of the same recipe to be processed in batches, or to save some idle time by processing a single job right away for a machine that can process batched jobs becomes a very complicated and difficult decision to make.

As shown in Fig. 1, there are N jobs that will undergo two stages of machines. The machines of the first stage conduct serial jobs, where each machine processes at most one job at a time. The machines of the second stage conduct batch jobs, where each machine can process at most two jobs of the same recipe simultaneously. Each job i with release time r_i and due date d_i that requires recipe j has to be assigned to a machine $m \in M_i$ in the first stage, and then be put in a batch k to a machine $m' \in M'_j$ in the second stage. These jobs should not stay in stage 2 for duration longer than Q_i. Our objective is to decide the best job-machine (which position for placing a job in which machine), job-batch (which job to which batch), and batch-machine (which batch position in which machine) assignments for each job, so that the makespan is minimized, and the workload (defined as the total processing time) for each machine is as balanced as possible.

A mixed integer programming model

Here we construct an MIP model for scheduling the two-stage FSMP problems. The notations of our model are listed below.

Indices, object sets, and parameters used in the model:

i:	index of job, $i = 1, \ldots, N$
m:	index of machine at the first stage, $m = 1, \ldots, M$
m':	index of machine at the second stage, $m' = 1, \ldots, M$
p:	index of position to place a job on a machine
k:	index of batch, $k = 1, \ldots, K$, where $K \leq N$
j:	index of recipe
I_m:	set of jobs that can be processed on machine m
M_{ji}:	set of machines capable of processing job i at the first stage
M'_j:	set of machines capable of processing recipe j at the second stage
I_j:	set of jobs that require recipe j
I^k:	set of jobs that can be included in batch k
B_j:	set of batches that require recipe j
B^i:	set of batches that can include job i
$P_{i,m}$:	processing time of job i on machine m
$P'_{k,m'}$:	processing time of batch k on machine m'
$t_{i,m}$:	transportation time for job i to machine m
$t'_{i,m}$:	transportation time for job i to machine m'
r_i:	release time of job i at the first stage
d_i:	due date of job i at the first stage
MT_m:	available time for machine m
$MT'_{m'}$:	available time for machine m'
Q_i:	the maximum waiting time of job i at the second stage
L:	a very large number to represent infinity

Decision variables in the model:

$X_{i,m,p} = 1$	if job i is assigned to position p on machine m; 0 otherwise
$Y_{i,k} = 1$	if job i is in batch k; 0 otherwise
$Y'_{k,m',p} = 1$	if batch k is assigned to position p on machine m'; 0 otherwise
$Z_{i,k,m',p} = 1$	if job i is in batch k which is assigned to position p on machine m'; 0 otherwise
$S_{m,p}$:	starting time of position p on machine m
$C_{m,p}$:	finishing time of position p on machine m
$S'_{m',p}$:	starting time of position p on machine m'
$C'_{m',p}$:	finishing time of position p on machine m'
r'_i:	the arrival time of job i at the second stage
d'_i:	the due date of job i at the second stage
$C_{\max}$:	makespan

Objective:

$$\text{Minimize } C_{\max} \tag{MIP}$$

Constraints:

$$\sum_{m \in M_i} \sum_{p} X_{i,m,p} = 1 \quad \forall i \tag{1}$$

$$\sum_{i\in I_m} X_{i,m,p} \le 1 \quad \forall m,\ p \tag{2}$$

$$\sum_{i\in I_m} X_{i,m,p} \ge \sum_{i\in I_m} X_{i,m,p+1} \quad \forall m,\ p \tag{3}$$

$$S_{m,p} + \sum_{i\in I_m} p_{i,m} X_{i,m,p} \le C_{m,p} \quad \forall m,\ p \tag{4}$$

$$C_{m,p} \le S_{m,p+1} \quad \forall m,\ p \tag{5}$$

$$MT_m \le S_{m,1} \quad \forall m \tag{6}$$

$$\mathrm{L}\cdot X_{i,m,p} + r_i + t_{i,m} - S_{m,p} \le \mathrm{L} \quad \forall i,m,p \tag{7}$$

$$\mathrm{L}\cdot X_{i,m,p} - d_i + C_{m,p} \le \mathrm{L} \quad \forall i,m,p \tag{8}$$

$$\mathrm{L}\cdot X_{i,m,p} - r'_i + C_{m,p} = \mathrm{L} \quad \forall m,\ p \tag{9}$$

$$r'_i + Q_i = d'_i \quad \forall i \tag{10}$$

$$\sum_{k\in B^i} Y_{i,k} = 1 \quad \forall i \tag{11}$$

$$\sum_{i\in I^k} Y_{i,k} \le 2 \quad \forall k \tag{12}$$

$$\sum_{m'\in M'_j} \sum_{p} Y'_{k,m',p} = 1 \quad \forall k \tag{13}$$

$$\sum_{k\in B^i} Y'_{k,m',p} \le 1 \quad \forall m',\ p \tag{14}$$

$$\sum_{k\in B^i} Y'_{k,m',p} \ge \sum_{k\in B^i} Y'_{k,m',p+1} \quad \forall m',\ p \tag{15}$$

$$Z_{i,k,m',p} \ge Y_{i,k} + Y'_{k,m',p} - 1 \quad \forall k \in B^i, i, m',\ p \tag{16}$$

$$S'_{m',p'} + \sum_{k\in B^i} p'_{k,m'} Z_{i,k,m',p} \le C'_{m',p} \quad \forall m',\ p \tag{17}$$

$$C'_{m',p} \le S'_{m',p+1} \quad \forall m',\ p \tag{18}$$

$$MT'_{m'} \le S'_{m',1} \quad \forall m' \tag{19}$$

$$\mathrm{L}\cdot Z_{i,k,m',p} + r'_i + t_{i,m'} - S'_{m',p} \le \mathrm{L} \quad \forall i,k,m',p \tag{20}$$

$$\mathrm{L}\cdot Z_{i,k,m',p} - d'_i + C'_{m',p} \le \mathrm{L} \quad \forall i,k,m',p \tag{21}$$

$$C_{\max} \ge C'_{m',p} \quad \forall m',\ p \tag{22}$$

$X, Y, Y', Z \in \{0, 1\}$, all other variables are nonnegative real numbers

This formulation contains $N^2(NM+2M+1)$ binary variables, $2N(2M+1)+1$ real variables and $M(3N^3+4N^2+8N+2)+5N$ constrains. Constraints (1)–(9) defines job assignments in stage 1, while constraints (10)–(22) defines the job and batch assignments in stage 2. In particular, constraint (1) arranges a job in some position (i.e. an order in the sequence of a job queue) on some machine; constraint (2) defines that each position in a machine can only be assigned at most one job; constraint (3) ensures jobs to be assigned on consequent positions on a machine. In other words, if position p on a machine m is empty, all of its latter positions have to be empty; constraint (4) and (5) respectively define the relations between the completion and starting times for each position p and its latter position $p+1$; constraint (6) limits the starting time MT_m for each machine m to be earlier than its first assigned job (i.e. $S_{m,1}$); constraint (7) defines the starting time to process job i in position p on machine m has to be earlier than its release time plus its transportation time; constraint (8) defines the completion time $C_{m,p}$ for processing job i in position p on machine m to be earlier than its due date, and that completion time becomes its release time r'_i in stage 2 by constraint (9); the due date of job i in stage 2 is defined by constraint (10); constraint (11) assigns a job in a batch, while each batch contains at most two jobs of the same recipe by constraint (12) and that batch has to be assigned by constraint (13); constraint (14) defines that each position in a machine of stage 2 can only be assigned at most one batch; constraint (15) ensures batches to be assigned on consequent positions on a machine. In other words, if position p on a machine m' is empty, all of its latter positions have to be empty; constraint (16) defines the situation of assigning job i to batch k in position p of machine m' in stage 2; constraints (17)–(21) are similar to constraints (4)–(8) of stage 1, from the viewpoint of assigning batch k in position p on machine m' of stage 2; constraint (22) defines makespan $C_{\max}$.

Although this MIP model can calculate an exact optimal schedule, it can only deal with problems of small size. For example, within 10 min, optimal schedules can only be calculated for a problem with size up to eight jobs on 2 machines at each stage, and no feasible solutions can be calculated for problems of more than 24 jobs on 6 machines at each stage. Such inefficiency is mainly caused by the enormous branch-and-bound iterations of solving MIP, which is exponential to problem size. Therefore, by reducing the size of an MIP, one can solves larger problems within shorter time.

A size-reduction heuristic

Here we propose a problem-reduction technique for MIP as shown in Fig. 2, based on the following observations: (1) machines capable of processing the same recipes are identical, (2) an optimal schedule usually averages the workload (defined as the sum of processing time) on identical machines, (3) we assume the workload is high, which means jobs are much more than machines and most machines should be busy without having idle time (this is generally true in semiconductor manufacturing industry), and (4) most of the NM positions in stage 1 are empty in the formulation. Note that the number of positions K is usually set to be N, the number of jobs, for general scheduling problems. Such a setting is too conservative since it considers putting all the jobs in a single machine, which rarely takes place in practice. In reality, each job is usually processable on more than one machine, so that no machine will be idle as long as there exists a waited job that it can process. In other words, if one can give better (i.e. smaller) upper bound estimation for K,

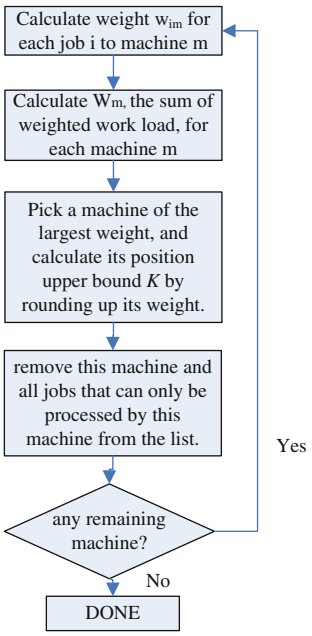

Fig. 2 Flow chart for our MIP size-reduction heuristics

the size of MIP can be reduced which in turn helps the MIP to solve larger problems in shorter time. To this end, we analyze the case when all jobs have zero release time since K in that case will have the largest possible value.

Constructing a bipartite graph $G = (N, A)$ where the node set $N = N_J \cup N_M$, and arc set $A = \{(i, m) : \text{job } i \in N_J, \text{ machine } m \in M_i \subseteq N_M\}$. In other words, for each machine m capable of processing job i, we connect job i to machine m by an arc (i, m) with weight $w_{im} = 1/|M_i|$. Then we calculate $W_m = \sum_{i \in I_m} w_{im}$, select the machine $m^* = \arg\max\{W_m\}$, then set the upper bound for its position, denoted by K_{m^*}, to be $\lceil W_{m^*} \rceil$. We then remove node m^* and its adjacent arcs, and repeat the above procedures to calculate K_m for $m = 1, \ldots, M$. Figure 2 illustrates the steps of our MIP size-reduction heuristic. The intuition of this size-reduction heuristic is based on workload balancing that is a common practice in real-world semiconductor manufacturing. By fairly distributing the jobs to the machines, we can effectively reduce K to a more reasonable value, which in turn helps to solve larger MIPS within shorter time.

The result of our computational experiments indicates this heuristic does serve its purpose. For example, within 10 min, it can calculate optimal schedules for cases with size up to $M = 4$, $N = 16$ and give feasible schedules for the cases of $M = 10$, $N = 48$, while the original MIP formulation can only calculate optimal schedule for the case of $M = 2$, $N = 8$ and give feasible schedules for the case of $M = 6$, $N = 24$. Nevertheless, such an improvement still could not solve some large-scale real-world problems (e.g. $M = 40$, $N = 160$) in semiconductor manufacturing industry. Hence, we further propose several heuristic approaches in next section.

Heuristics based on dispatching rules and local search techniques

To meet the real-world scheduling requirement in semiconductor manufacturing industry, our goal is to schedule 160 jobs on 40 machines at each stage within 10 min so that its makespan is as short as possible. Based on previous experiences in solving MIPs, we have learned the following two facts: (1) the mathematical programming approaches that solve MIPs are usually very time-consuming, and (2) the optimal first stage job-machine assignment based on MIP may not help too much in shortening the makespan for the second stage batch-machine assignment. This is because a good stage 1 schedule and an optimal stage 1 schedule may both give the same or similar stage 2 schedule, if the stage 2 scheduling is carefully handled. As a result, we propose to first solve the stage 1 schedule by dispatching rules, and then solve the stage 2 schedule either by MIP (see section "Combination of BFIFO and MIP model") or by local search heuristics (see section "Combination of BFIFO and ITT").

In particular, two dispatching rules: FIFO (First In First Out) and BFIFO (Batch First In First Out) are proposed. FIFO assigns position for a job according to its arrival time. BFIFO uses FIFO in the first stage, and then groups two jobs of the same recipe according to their arrival time in the second stage. For example, to assign n' jobs where n' is an odd number, we will test two different grouping mechanisms: (1,2), (3,4),…, $(n'-2, n'-1)$, (n'), or (1), (2,3),…, $(n'-1, n')$, and then pick the more efficient one. After assigning all jobs to batches, we schedule these batch jobs based on the FIFO rule. By this way, we can avoid long waiting time between two assigned jobs. The BFIFO flow chart is shown in Fig. 3.

From the results of our computational experiments (see section "Computational experiments and analyses"), BFIFO performs better than FIFO. Therefore, we suggest to first use the scheduling decision by BFIFO as a start, then we apply the following two approaches to further improve the solutions: (1) mixed integer programming model (MIP); and (2) local search techniques based on Interchange, Translocation, and Transposition (ITT) mechanisms.

Combination of BFIFO and MIP model

To overcome the disadvantages of conventional MIP models, this approach search for a good solution by MIP in a reduced solution space obtained by BFIFO. In particular, after dispatching the jobs and batches by BFIFO, we calculate a new and better stage 2 schedule by an MIP model, based on the BFIFO stage 1 schedule. The advantage of this technique is to

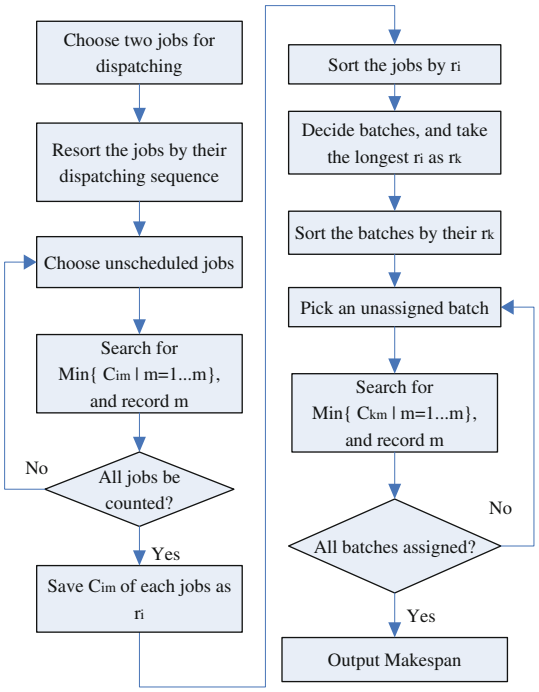

Fig. 3 Flow chart of BFIFO

use MIP to search for the optimal or better stage 2 schedules in the reduced solution space defined by the BFIFO stage 1 schedule. The objective and constraints are as follows:

Objective:

$$\text{Minimize } C_{\max} \qquad \text{(BFIFO + MIP)}$$

Constraints:

$$r'_k + Q_k = d'_k \quad \forall k \tag{23}$$

$$\sum_{m' \in M'_j} \sum_{p} Y'_{k,m',p} = 1 \quad \forall k \tag{24}$$

$$\sum_{k \in B^i} Y'_{k,m',p} \le 1 \quad \forall m', p \tag{25}$$

$$\sum_{k \in B^i} Y'_{k,m',p} \ge \sum_{k \in B^i} Y'_{k,m',p+1} \quad \forall m', p \tag{26}$$

$$S'_{m',p'} + \sum_{k \in B^i} p'_{k,m'} Y'_{k,m',p} \le C'_{m',p} \quad \forall m', p \tag{27}$$

$$C'_{m',p} \le S'_{m',p+1} \quad \forall m', p \tag{28}$$

$$MT'_{m'} \le S'_{m',1} \quad \forall m' \tag{29}$$

$$\mathrm{L} \cdot Y'_{k,m',p} + r'_i + t_{i,m'} - S'_{m',p} \le \mathrm{L} \quad \forall i, k, m', p \tag{30}$$

$$\mathrm{L} \cdot Y'_{k,m',p} - d'_i + C'_{m',p} \le \mathrm{L} \quad \forall i, k, m', p \tag{31}$$

$$C_{\max} \ge C'_{m',p} \quad \forall m', p \tag{32}$$

$Y' \in \{0, 1\}$, all other variables are nonnegative real numbers

Since the stage 1 schedule is already determined by BFIFO, we only need to calculate the optimal batch-machine assignments in stage 2. Therefore the BFIFO+MIP model can be viewed as a subproblem of the original MIP model and share similar constraints. In particular, constraint (23) corresponds to constraint (10); constraints (24)–(26) are identical to constraints (13)–(15); and constraints (27)–(33) are mapped from constraints (17)–(22).

This heuristic does help to boost the efficiency of MIP and improve the effectiveness of BFIFO. In our testings, we find that this heuristic may find optimal solutions for small scale problems (16 or 32 jobs) within 10 min, but it still can not deal with problems with size up to 96 jobs. We can have a better solution than BFIFO for the cases with size up to 112 jobs, but still can not solve the problems containing more than 160 jobs.

To further speed up the solution procedures, in next section we try to conduct local search techniques which insert or interchange jobs between or within machines whenever the local search operations take to shorten the makespan.

Combination of BFIFO and ITT

Although BFIFO tends to assign a job to a machine in a greedy fashion (e.g. a machine that requires shorter processing time), it gives no guarantee in the solution quality. Nevertheless, based on the current BFIFO scheduling decision, local search techniques that conduct job insertions or interchanges within or between machines can help improve the solution quality very efficiently. To this end, we propose three solution improving mechanisms: Interchange, Translocation, and Transposition, together denoted by ITT.

Interchange is to swap jobs on different machines as shown in Fig. 4. Translocation is to pick a job from the machine with the longest processing time, and then insert the selected job to another machine. Here we test two translocation techniques: (1) inserting a job according to BFIFO order (see Fig. 5), and (2) inserting a job to each possible position (see Fig. 6). Obviously the solution given by the second Translocation technique should be at least as good as

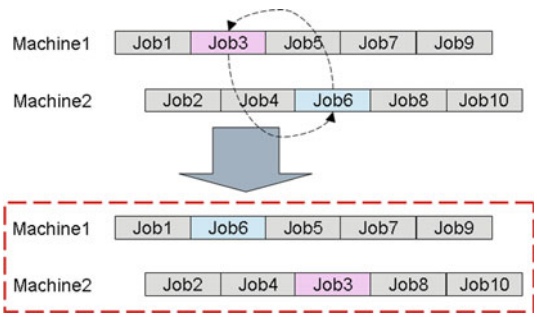

Fig. 4 Interchange mechanism

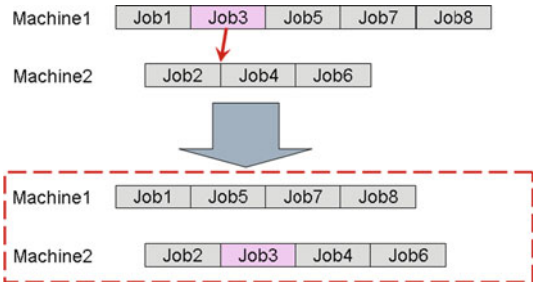

Fig. 5 Translocation mechanism based on BFIFO order

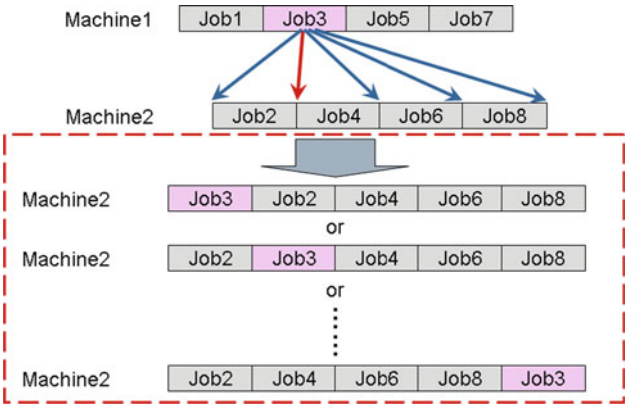

Fig. 6 Translocation mechanism for all possible positions

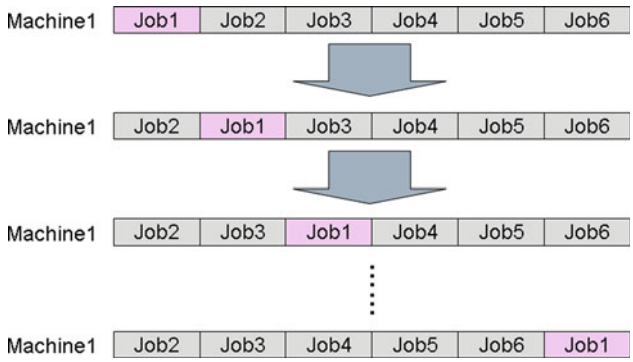

Fig. 7 Transposition mechanism

the one by the first Translocation technique, but it takes much more time. In our testings, the first Translocation technique is much faster and very often it also gives the same solution as the second Translocation technique.

Transposition is to swap jobs on the same machine as shown in the Fig. 7. Conducting these mechanisms in different orders may give different levels of improvement. In our experience, iteratively conducting Interchange and Translocation mechanisms until no further improvement and then conduct the Transposition mechanism seems to be more effective, and thus we use these settings as our default ITT procedures.

Computational experiments and analyses

Our MIP formulations are solved by ILOG CPLEX 11.1.1. All other heuristic algorithms (including BFIFO and ITT mechanism) are written in C++. All the tests are conducted on a personal computer with Intel(R) Pentium(R) D CPU 3.00 GHz and 1.5 GB RAM with the Windows XP operating system.

We have tested seven solution methods to our problems: MIP, MIPH, FIFO, BFIFO, BFIFO+MIP, BFIFO+MIPH and BFIFO+ITT. In particular, MIP and MIPH respectively represent the mixed integer programming models of full and reduced size, as described in section "A mixed integer programming model" and "A size-reduction heuristic"; BFIFO is the proposed dispatching rule considering batch jobs, as introduced in section "Heuristics based on dispatching rules and local search techniques"; BFIFO+MIP and BFIFO+MIPH, described in section "Combination of BFIFO and MIP model", are the integration of BFIFO with MIP and MIPH, respectively; and BFIFO+ITT denotes the heuristic in section "Combination of BFIFO and ITT" that iteratively improves the solution of BFIFO by the ITT mechanism.

The parameters to setup our random test cases are generated based on the manufacturing data of a semiconductor manufacturing company in Taiwan. In stage 1, the available time of machines is a random number uniformly generated from the interval [10, 20] (denoted by U(10,20) afterwards); the job arrival time follows a distribution of U(0,62.5); the job setup time on the machines follows a U(1,10) distribution; and the processing time of machines has a U(20,30) distribution. In stage 2, the available time of machines has a U(20,40) distribution; the job setup time on the machines follows a U(1,10) distribution; the processing time has a U(30,40) distribution; and the number of jobs is approximately four time of the number of machines.

We have tested twenty sets of problems in different scales. Each test set is composed by 10 randomly generated test data with different random parameters. The result of different approaches are compared and illustrated by line charts, as shown in Figs. 8, 9, 10, 11, 12 and 13.

The results indicate that MIPH can not solve cases of more than 48 jobs within 10 min, so we use MIP and MIPH to test the cases from 8 jobs to 48 jobs. Figure 8 shows that MIP and MIPH are very time-consuming, although MIPH has better efficiency and is capable of dealing with large cases. On the other hand, all other heuristic approaches are much faster than these two.

As to the quality of solutions by different solution methods, Fig. 9 shows that most heuristic approaches have better and more stable solutions for the cases of 8–48 jobs, although they are not guaranteed to be optimal. On the other hand, although MIP and MIPH can find optimal solutions, they are only suitable for small-scale problems and the quality of their

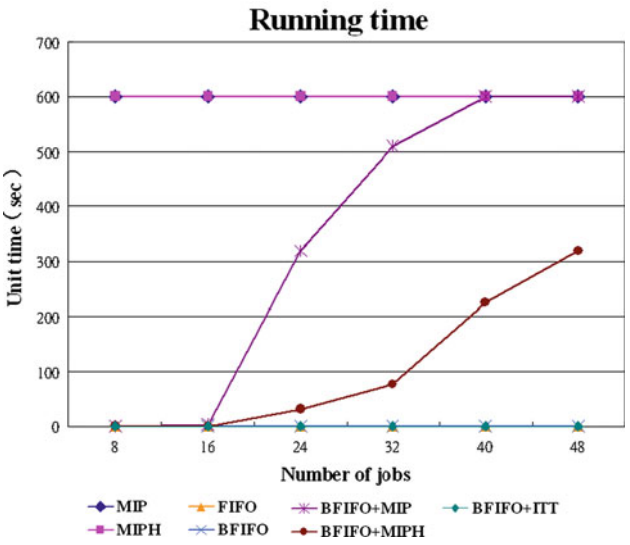

Fig. 8 Running time of seven solution methods

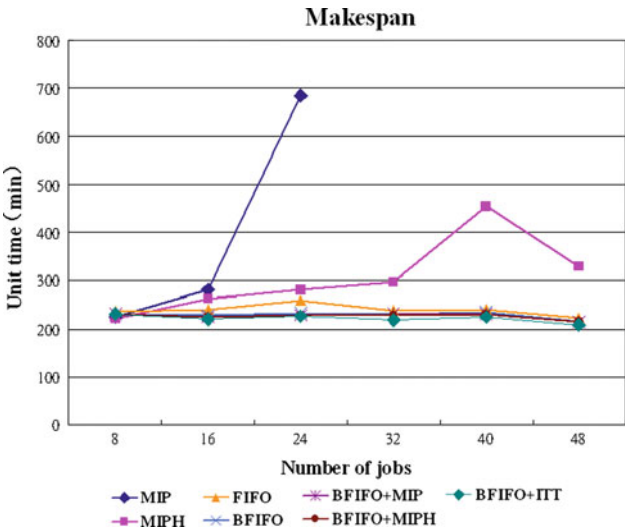

Fig. 9 Makespan of seven solution methods

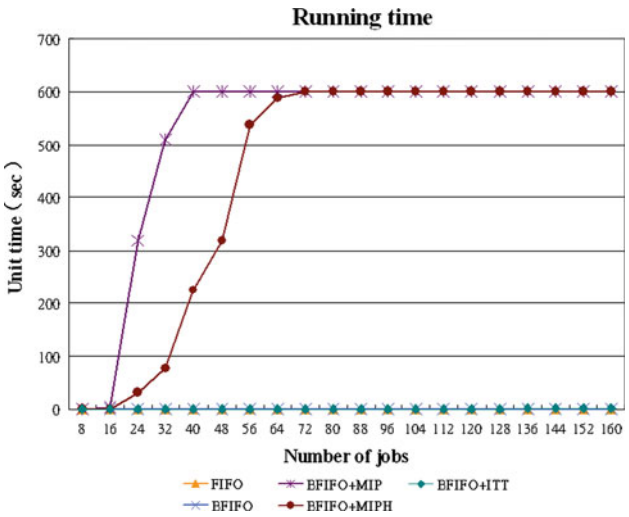

Fig. 10 Running time of five heuristic approached

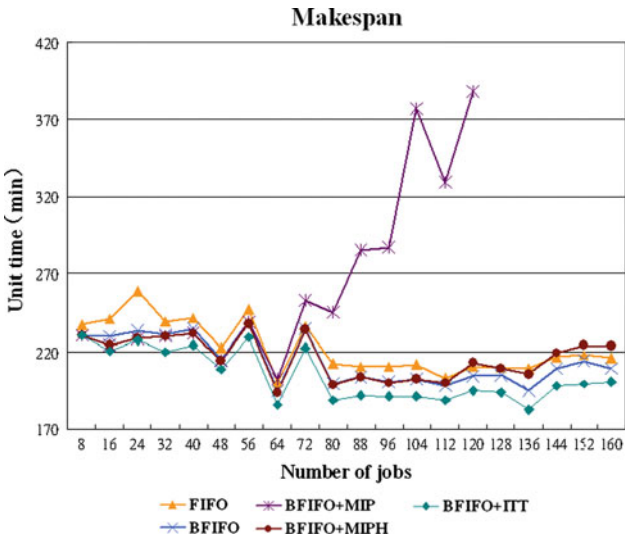

Fig. 11 Makespan of five heuristic approaches

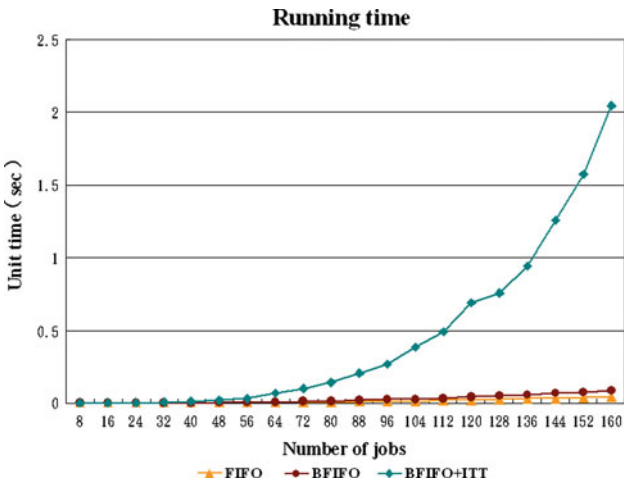

Fig. 12 Running time of FIFO, BFIFO and BFIFO+ITT

feasible solutions drop down very fast as the number of jobs increase, as shown in Fig. 10.

For solving large-scale problems, we can only rely on the five heuristic approaches (i.e. FIFO, BFIFO, BFIFO+MIP, BFIFO+MIPH, and BFIFO+ITT) to calculate solutions of acceptable qualities. Figure 11 indicates that BFIFO+ITT gives solutions of the best qualities among all the five heuristic approaches.

Now we further compare and analyze the performance of the three most efficient heuristic approaches: FIFO, BFIFO, and BFIFO+ITT. Figure 13 indicates that ITT mechanism does serve its purpose by effectively reducing the makespan without sacrificing the efficiency (as shown in Fig. 12). In particular, BFIFO+ITT usually gives the best solutions in all

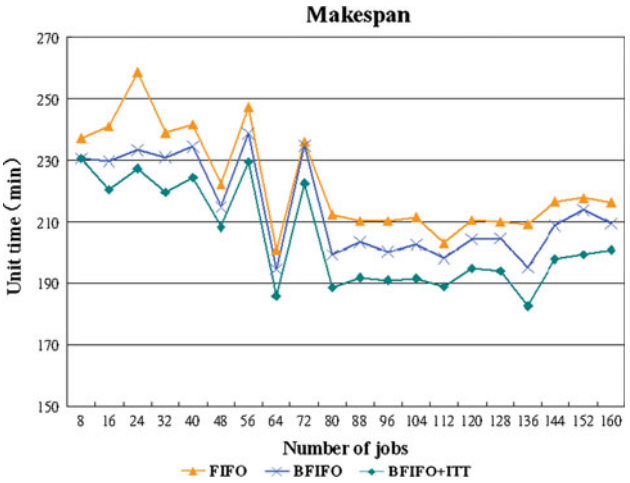

Fig. 13 Makespan of FIFO, BFIFO, and BFIFO+ITT

of our tests, and it improves the quality of BFIFO solutions by 5–10% for cases of $N = 160$ and $M = 40$ within 3 s.

In summary, we have the following three observations from our computational experiments: (1) all the mathematical programming based approaches are time-consuming, and thus are only suitable for cases of small scale (e.g. up to 16 jobs on 4 machines at each stage); (2) among the two proposed mathematical programming based approaches, MIPH usually performs better than MIP in both efficiency and effectiveness. However, in scheduling fewer than 16 jobs and when the machines have very different processing time over the same recipes, MIPH may give worse solutions than MIP since the assumption of identical machines no longer holds which makes the upper bound estimation of positions become inaccurate and spoils the solution qualities of MIPH. For the cases of 16–48 jobs, MIPH consistently beats MIP. Within 10 min, MIPH can schedule up to 48 jobs on 12 machines at each stage, while MIP can at most schedule 24 jobs on 6 machines at each stage; and (3) among those five heuristic approaches, BFIFO always give better solutions than the traditional FIFO. By integrating MIP or MIPH with BFIFO, solutions better than the one by BFIFO alone can be calculated for small scale problems (less than 64 jobs). BFIFO+MIPH always beats BFIFO+MIP for larger cases (e.g. more than 64 jobs). Overall, BFIFO+ITT, the heuristic that integrates the ITT mechanism to BFIFO, has the best performance that can solve all the test problems within 3 min.

Conclusions

In order to deal with complex semiconductor manufacturing processes that involve two stages of job-machine, job-batch, and batch-machine assignments with release time and due dates, this paper proposes several solution methods including the conventional MIP models, effective dispatching rules that take the batch jobs into consideration, and local search mechanisms that insert and swap jobs to positions within or between machines. Our MIP problem-reduction technique (MIPH) is designed based on the common practices in real-world semiconductor manufacturing industry that balance workload among machines. Although MIPH is more efficient and effective than the conventional MIP for most cases, it still cannot deal with larger cases. Furthermore, we observe that any solution method that involves MIP or MIPH will still require much running time for larger cases, and thus we conclude those MIP based approaches are not suitable for some real-world two-stage scheduling problems that require to assign 160 jobs to 40 machines at each stage within 10 min. In other words, only FIFO, BFIFO, and BFIFO+ITT can meet the real-world dynamic production planning requirements in semiconductor manufacturing industry.

Our computational experiments indicate that BFIFO, the heuristic considering batch job assignment with FIFO dispatching rule, consistently beats FIFO. When integrating BFIFO with our proposed ITT local search mechanism, the ITT heuristic can further effectively improve the quality and robustness for a BFIFO solution within a very short time interval (e.g. 3 min). As a result, we recommend the use of BFIFO+ITT to solve difficult scheduling problems in real-world semiconductor manufacturing. Although our proposed heuristics aim to minimize the makespan, with minor modification it can also deal with other objectives. For example, to minimize the tardiness caused by violation on the due date requirements, one can first calculate a better initial scheduling based on the earliest due date (EDD) rule. Then, in the local search part, one just checks whether any interchange, translocation, or transposition operation leads to less tardiness.

For future research, we also suggest investigations on the performance of our proposed techniques over other difficult real-world scheduling problems. A more thorough numerical analysis on the performance comparison between other sophisticated heuristics to our proposed methods would also be interesting and useful.

Acknowledgments I.-Lin Wang is partly supported by the National Science Council of Taiwan under Grant NSC 98-2410-H-006-115-MY2.

References

Bellanger, A., & Oulamara, A. (2009). Scheduling hybrid flowshop with parallel batching machines and compatibilities. *Computers and Operations Research, 36*, 982–1992.

Centeno, G., & Armacost, R. L. (2004). Minimizing makespan on parallel machines with release time and machine eligibility restrictions. *International Journal of Production Research, 42*, 1243–1256.

Dobson, G., & Nambimadom, R. S. (2001). The batch loading and scheduling problem. *Operations Research, 49*, 52–65.

Grangeon, N., Tanguy, A., & Tchernev, N. (1999). Generic simulation model for hybrid flow-shop. *Computers and Industrial Engineering, 37*, 207–210.

Hung, Y. F. (1998). Scheduling of mask shop E-beam writers. *IEEE Transactions on Semiconductor Manufacturing, 11*, 165–172.

Hunsucker, J. L., & Shah, J. R. (1994). Comparative performance analysis of priority rules in a constrained flow shop with multiple processors environment. *European Journal of Operational Research, 72*, 102–114.

Kashan, A. H., Karimi, B., & Jenabi, M. (2008). A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes. *Computers and Operations Research, 35*, 1084–1098.

Kuo, Y., Yang, T., Peters, B. A., & Chang, I. (2007). Simulation metamodel development using uniform design and neural networks for automated material handling systems in semiconductor wafer fabrication. *Simulation Modeling Practice and Theory, 15*, 1002–1015.

Li, C. L., & Lee, C. Y. (1997). Scheduling with agreeable release times and due dates on a batch processing machine. *European Journal of Operational Research, 96*, 564–569.

Malve, S., & Uzsoy, R. (2007). A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families. *Computer and Operations Research, 34*, 3016–3028.

Mathirajan, M., & Sivakumar, A. I. (2006). A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *International Journal of Advanced Manufacturing Technology, 29*, 990–1001.

Petroni, A., & Rizzi, A. (2002). A fuzzy logic based methodology to rank shop floor dispatch rules. *International Journal of Production Economics, 76*, 99–108.

Potts, C. N. & Kovalyov M. Y. (2000). Scheduling with batching: A review. *European Journal of Operational Research, 120*, 228–249.

Uzsoy, R., Church, L. K., & Ovacik, I. M. (1992). Dispatching rules for semiconductor testing operations: A computational study. *IEEE/CHMT International Electronic Manufacturing Technology Symposium* (pp. 272–276).