International Journal of Production Research
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tprs20

Scheduling semiconductor in-line steppers in new product/process introduction scenarios

Muh-Cherng Wu [a] & Chie-Wun Chiou [a]

[a] Department of Industrial Engineering and Management , National Chiao Tung University , Hsin-Chu, 300 Taiwan, ROC

Published online: 23 Feb 2009.

To cite this article: Muh-Cherng Wu & Chie-Wun Chiou (2010) Scheduling semiconductor in-line steppers in new product/process introduction scenarios, International Journal of Production Research, 48:6, 1835-1852, DOI: 10.1080/00207540802577920

To link to this article: http://dx.doi.org/10.1080/00207540802577920

International Journal of Production Research
Vol. 48, No. 6, 15 March 2010, 1835–1852

Scheduling semiconductor in-line steppers in new product/process introduction scenarios

Muh-Cherng Wu* and Chie-Wun Chiou

Department of Industrial Engineering and Management, National Chiao Tung University, Hsin-Chu, 300 Taiwan, ROC

(*Received 10 December 2007; final version received 17 October 2008*)

This paper presents a scheduling method for an in-line stepper operating in a new process/production introduction (NPI) scenario. An in-line stepper is a bottleneck machine in a semiconductor fab. Its interior is comprised of a series of chambers, while its exterior is a *dock* equipped with a limited number of ports. The transportation unit for each chamber is a piece of wafer, while that for each port is a job that can contain up to 25 wafers. This transportation incompatibility may lead to an unexpected capacity loss for an in-line stepper – in particular in an NPI scenario that, by nature, includes a substantial number of *small-sized* jobs. Such a capacity loss can be alleviated by effective scheduling. A genetic algorithm (GA) scheduling method is proposed to enhance the productivity of in-line steppers. Four other sequencing methods are compared with the GA method. Numeric experiments indicate that the GA method outperforms the four benchmarks. The higher the percentage of small-sized jobs, the better the performance of the GA.

Keywords: activity based costing; design of experiments; design for manufacture; availability; capability indices; computer vision; control charts; robotics

1. Introduction

Semiconductor manufacturing is a capital-intensive industry. Building a semiconductor factory (also called a fab) for manufacturing integrated circuits on 12 inch wafers costs about 2.5 billion dollars, 80% of which is costs for tool acquisition. Among the tools, a particular tool type, the stepper, is the most expensive and relatively limited in quantity. Therefore, steppers are usually the bottleneck of a wafer fab, and their utilisation would significantly affect the ultimate throughput of a fab.

One way to effectively utilise a stepper is by appropriately dispatching the wafer jobs (also called lots) that are waiting before it. Job dispatching denotes which job should be processed first while a stepper is available. There is a substantial literature on stepper dispatching (Chern and Liu 2003, Díaz *et al*. 2005, Miwa *et al*. 2005) that has established significant milestones.

Most prior studies on stepper scheduling are implicitly based on two assumptions. First, the production yield is quite high; each wafer job to be scheduled is a *full lot* (a typical full lot carries 25 wafers). Second, a stepper is interfaced with a transport mechanism of *infinite capacity*. That is, a new job can always access the stepper as long as it is not fully utilised.

*Corresponding author. Email: mcwu@mail.nctu.edu.tw

ISSN 0020–7543 print/ISSN 1366–588X online
© 2010 Taylor & Francis
DOI: 10.1080/00207540802577920
http://www.informaworld.com

However, a wafer job may not always be full sized for two reasons. Firstly, engineers tend to form a small-sized job for process recipe justification in the new product/process introduction (NPI) stage. Secondly, in the pilot run stage, the yield can be very low and uncertain. Small-sized wafer jobs can be quite substantial – in particular for a foundry fab, which produces customised products and has hundreds of customers to serve and thousands of products to produce.

Moreover, the inclusion of a substantial number of small-sized jobs may overturn the infinite capacity assumption on the transport mechanism for steppers. That is, in a low-yield scenario, a new job may not be able to access an underutilised stepper due to the capacity limit of its transport mechanism. This limitation would not usually appear in a high-yield scenario and has rarely been observed.

This paper attempts to develop a scheduling algorithm for *a relatively advanced version of steppers* (called in-line steppers) in a low-yield environment. An in-line stepper is composed of a series of chambers, equipped with an entry transport mechanism. This problem, therefore, can also be regarded as a special type of flow-shop scheduling problem with certain salient features.

The remainder of the paper is organised as follows. Section 2 reviews the scheduling literature on flow shops and steppers. Section 3 describes the operational mechanism of an in-line stepper. Section 4 presents an abstraction model for in-line steppers and describes how to compute the makespan of a job sequence. A genetic algorithm (GA) for solving the scheduling problem is presented in Section 5. Our experimental results for the GA and its four benchmarks are reported in Section 6. The concluding remarks are presented in the last section.

2. Literature review

The in-line stepper is essentially a flow shop when considering each type of chamber as a workstation. We therefore review flow shop scheduling as well as job sequencing on steppers.

Flow shop scheduling problems have been extensively studied, and there are some survey papers that are available (Reisman *et al.* 1997, Allahverdi *et al.* 1999, Yang and Liao 1999, Cheng *et al.* 2000, Pinedo 2000, Hejazi and Saghafian 2005, Bagchi 2006, Zhu and Wilhelm 2006, Quadt and Kuhn 2007). The uniqueness of a flow shop scheduling problem can be characterised by the scheme $(\alpha|\beta|\gamma)$ (Graham *et al.* 1979), where α denotes types of workstations, β denotes process constraints, and γ denotes the objective function.

Process constraints essentially address the interface between two subsequent workstations. Some example constraints include buffer size limitation (Pranzo 2001), maximum allowable waiting time after exiting a workstation (Sriskandarajah 1993), and setup time for a workstation (Danneberg *et al.* 1998). The port constraint addressed in our problem is a special type of buffer size limitation; however, it has scarcely been examined in the literature.

Based on a comprehensive survey (Gupta and Sivakumar 2006), the solution methods for job scheduling in semiconductor manufacturing can be grouped into four categories: dispatching rules, artificial intelligence (AI) techniques, mathematical programming techniques, and meta-heuristic algorithms.

The dispatching rule approach attempts to identify an optimal dispatching rule for a production system through the use of discrete event simulation. Prominent dispatching rules (Panwalker and Iskander 1977, Dabbas and Fowler 2003) include first-in-first-out (FIFO), shortest processing time (SPT), largest processing time (LPT), and earliest operation due date (Operation-EDD). However, the optimal dispatching rule may vary among different production scenarios. Some studies have used AI techniques to *dynamically* identify an optimal dispatching rule based on the time-varying characteristics of a production system (Aytug *et al.* 1994).

In the category of mathematical programming techniques, the job scheduling problem has mostly been formulated as an integer program (IP) (Kang and Lee 2007). Due to the inherent complexity of an IP, the approach is more suitable for small-scale problems, that is, less than 20 jobs. In solving a large-scale problem, meta-heuristic algorithms such as a genetic algorithm (GA), tabu search, and simulated annealing have been widely used (Cavalieri *et al.* 1999, Wang and Uzsoy 2002).

Different from the track of developing new solution methodologies, some studies have claimed their uniqueness in terms of problem characteristics. That is, the stepper scheduling problem may vary due to the inclusion of different constraints imposed on the steppers – for example, mask setup (Chern and Liu 2003, Duwayri *et al.* 2006), machine dedication (Wu *et al.* 2006, 2008a, 2008b), rework (Sha *et al.* 2006), and cluster tool configuration (Morrison and Martin 2007). Nonetheless, these previous studies implicitly assumed that the production systems are in a *high-yield* status. Our stepper scheduling problem is unique because it addresses a scenario with a *low-yield* status, which mostly appears in the introduction of new processes/products.

3. Operational mechanism of in-line steppers

Figure 1 shows that the production system examined is composed of three subsystems: the input/output WIP buffers, the dock for job entry/departure, and the in-line stepper. The input/output WIP buffers, with infinite buffer size by assumption, are designed to store wafer jobs. Each job is a cassette (i.e. container) that carries 1–25 wafers. The dock is a limited-size buffer, which could accommodate up to four cassettes. The in-line stepper is a machine consisting of a group of chambers, each of which can process one wafer at a time. The transportation unit between the WIP buffers and the dock is a *job*, but that between the dock and the in-line stepper is a piece of *wafer*. A *transportation incompatibility* therefore exists between the dock and the in-line stepper.

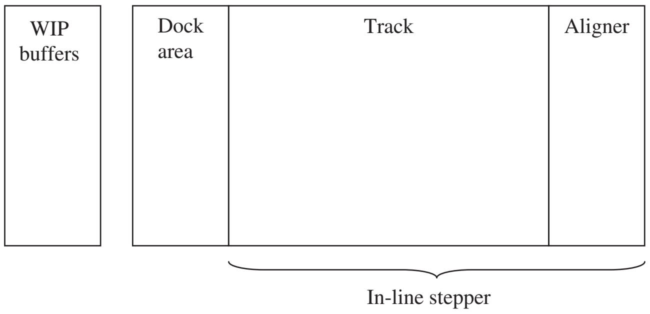

Figure 1. Production system of the stepper.

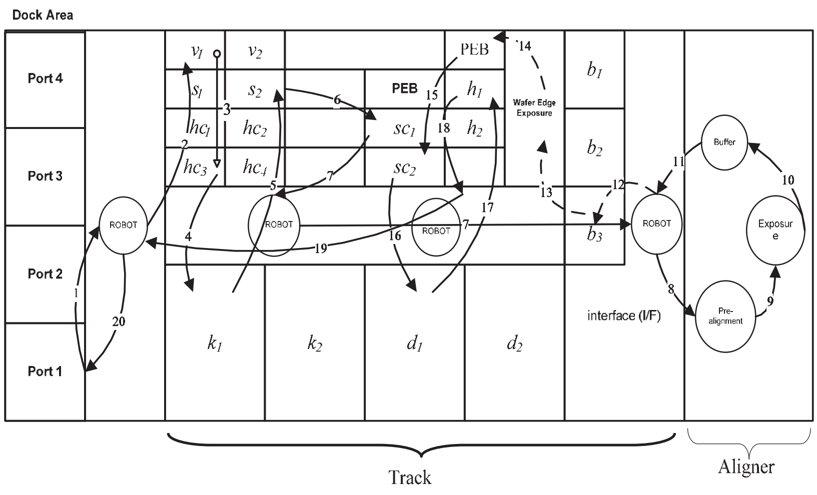

Figure 2. Chambers of an in-line stepper: vapor prime, v_1 and v_2; cooling, hc_1–hc_4; coater, k_1 and k_2; soft-bake, s_1 and s_2; cooling, sc_1 and sc_2; buffer, b_1–b_3; cooling, pc_1 and pc_2; develop, d_1 and d_2; hard bake, h_1 and h_2.

Figure 2 shows a detailed layout of a typical in-line stepper, which is composed of two modules – a *track* and an *aligner* (Quirk 2001, Xiao 2001, Zant 2004). The *track* module involves seven types of chambers, where each type may be *more than one* in number. The *aligner* module contains three types of chambers, where each type typically involves *only one* chamber.

The operation sequence for a wafer in the *in-line* stepper involves seven key *steps*. The detailed *moves* of these key steps are described below (see Figure 2).

Step 1: HMDS adhesion (***Moves 1–3***)

In Moves 1 and 2, a wafer is transported by a robot from its accommodation port to the *vapor prime chambe*r, where the wafer surface is cleaned, dehydrated, and primed with hexamethyldisilazane (HMDS) to enhance the adhesion between the photo-resist and the wafer surface. In Move 3, the wafer is transported to a *cooling* chamber to cool the wafer because the HMDS adhesion involves a high-temperature process. Note that the in-line stepper includes two vapor prime chambers (v_1 and v_2) and four cooling chambers (hc_1–hc_4).

Step 2: Photo-resist coating (***Move 4***)

In Move 4, the wafer is transported to a coater chamber, where liquid photo-resist is coated onto the wafer surface. In the in-line stepper, there are two coater chambers (k_1 and k_2).

Step 3: Soft-bake (***Moves 5–7***)

In Move 5, the wafer is transported to a soft-bake chamber to drive off the solvent in the photo-resist and consequently improve adhesion. In Move 6, the wafer is transported to a *cooling* chamber to cool the wafer. In Move 7, the wafer is transported to a robot. In the in-line stepper, there are two soft-bake chambers (s_1 and s_2), and two cooling chambers (sc_1 and sc_2).

***Step 4: Alignment* (*Moves 8–12*)**
In Move 8, the wafer is transported to the *pre-alignment* chamber, which subsequently attempts to place the wafer in an appropriate location and orientation. In Move 9, the wafer is transported to the aligner chamber to perform the exposure operation. Note that different exposure operations require different masks. That is, a *mask change* has to be made if the next exposure operation is different from the present one. In Move 10, the wafer is transported to the *buffer* chamber, which serves as a temporary storage for the waiting robot's transportation. In Moves 11 and 12, the wafer is transported to a buffer on the track. Note that the in-line steppers include three buffer chambers (b_1–b_3).

***Step 5: Post-exposure bake* (*Moves 13–15*)**
In Move 13, the wafer is transported to the *wafer edge exposure chamber* to remove the photo-resist on a perimeter ring area of the wafer surface. In Move 14, the wafer is transported to the post-exposure bake (PEB) chamber for baking the photo-resist; this is a high-temperature process. In Move 15, the wafer is transported to a cooling chamber. The in-line steppers include two cooling chambers (pc_1 and pc_2).

***Step 6: Develop* (*Move 16*)**
In Move 16, the wafer is transported to a *develop* chamber to selectively remove a part of the photo-resist on the wafer surface to display the circuit pattern. The in-line stepper includes two develop chambers (d_1 and d_2).

***Step 7: Hard-bake* (*Moves 17–19*)**
In Move 17, the wafer is transported to a hard-bake chamber to facilitate the evaporation of the solvent of the remaining photo-resist to improve the adhesion of the photo-resist onto the wafer surface. In Moves 18–20, the wafer is transported to its original accommodation port on the dock. The in-line stepper includes two hard-bake chambers (h_1 and h_2).

In practice, a wafer has to undergo about 20 different exposure operations before its completion. That is, the wafer has to enter the in-line steppers about 20 times. In each entry, the flow sequence is almost the same but may vary in processing times and may need different masks. A mask setup, therefore, may be needed at the exposure operation. With the same sequence of operations, an in–line stepper can therefore be regarded as a *flow shop* system with *parallel machines*; that is, an operation may be served by a group of chambers.

In the dock area, an in-line stepper typically involves four ports. Each port can accommodate only one wafer cassette. In processing its wafers, a cassette must stay at the port until all its wafers have been completed. This implies that no other job can access the in-line stepper when all the ports are occupied. As a result, a *capacity loss* caused by *port-capacity constraint* may appear in a scenario with substantial small-sized wafer jobs.

An example of such a *capacity loss* is given below. Consider an in-line stepper that has 22 chambers equipped with four ports, where *four* jobs (A, B, C, and D) are on the dock and *one* job (E) is off the dock waiting for a free port. Job A contains 25 wafers and jobs B, C, and D in total carry only 16 wafers. Suppose job A is processed first, followed by jobs B, C, and D. Each wafer of the four jobs in the sequence moves through the in-line stepper. The 16 wafers of B, C, and D follow the step of the last wafer in job A. When all the wafers of job A leave the stepper and move back to its cassette, the four ports still accommodate jobs A, B, C, and D. At this point, the wafers of jobs B, C, and D occupy the last 16 chambers of the stepper and the first six chambers have no wafers to host. Only after job A is moved away can job E then be hosted by the port and processed by the in-line stepper. Such a capacity loss of chambers is very critical because in-line steppers are often the bottlenecks of a fab.

In summary, the in-line stepper of interest can be interpreted as *a parallel-machine flow shop with four salient features: mask setup, port-capacity constraint, dock-stepper transportation incompatibility, and variable job size*. The four salient features as a whole have scarcely been considered in prior scheduling studies.

4. Makespan evaluation for job sequences

A method is presented for evaluating the *makespan* of a job sequence for the in-line stepper scheduling problem. The evaluation method, essentially a simulation approach, is adapted from Ruiz and Maroto (2006). That is, we virtually sent each wafer in the order following the job sequence into the in-line stepper and looked for an available chamber that can finish the job at the earliest time. In the simulation, setup times were considered at the aligner chamber.

The makespan evaluation procedure is presented below, where a group of functionally identical chambers is called a *stage*.

Notation

j	index of job
k	index of wafer
i	index of stage
l	index of chamber
a	index of the aligner chamber
ρ	total number of ports in the dock
n	total number of jobs to be processed by the in-line stepper
M	total number of stages in the in-line stepper
m_i	total number of chambers at stage i
p_{iljk}	processing time required for chamber l at stage i to process wafer k in job j
π	a job sequence for the n jobs, $\pi = [\pi(1), \ldots, \pi(n)]$
$\pi(j)$	the job in the jth position of sequence π
$w(j)$	total number of wafers in job j
t_{u}	transportation time for uploading a job to the dock
t_{d}	transportation time for downloading a job from the dock
$S_{i,l,\pi(j),k}$	setup time required for chamber l in stage i to process wafer k in job $\pi(j)$; if $i \neq \alpha$ or $k \neq 1$, then $S_{i,l,\pi(j),k} = 0$; otherwise $S_{i,j,\pi(j),k} = \delta_{\pi(\mathrm{j}),\pi(\mathrm{j}-1)}$
$\delta_{\pi(j),\pi(j-1)}$	setup time required for the aligner chamber to switch production from job $\pi(j-1)$ to job $\pi(j)$; $\delta_{\pi(j),\pi(j-1)} = s_0$ if $\pi(j-1)$ and $\pi(j)$ use different masks, and $\delta_{\pi(j),\pi(j-1)} = 0$ otherwise
$A_{i,l,t}$	the time epoch when chamber l in stage i just becomes available. That is, when chamber (i,l) is free at t, $A_{i,l,t}$ is the last

wafer-completion-epoch before t; when chamber (i,l) is in operation at t, $A_{i,l,t}$ is the *first wafer-completion-time* after t

$C_{i,\pi(j),k}$	completion time of wafer k in job $\pi(j)$ at stage i
$C_{\max}(\pi)$	makespan of job sequence π

The makespan evaluation procedure is governed by the following equations:

$$C_{i,\pi(j),k} = \min_{1\le l\le m_i} \{\max\{A_{i,l,t} + S_{i,l,\pi(j),k}, C_{i-1,\pi(j),k}\} + p_{i,l,\pi(j),k}\}, \quad \text{where } t = C_{i-1,\pi(j),k} \quad \text{for } 1 \le i \le M, \tag{1}$$

$$C_{M+1,\pi(j),w(\pi(j))} = C_{M,\pi(j),w(\pi(j))} + t_{\mathrm{d}}, \tag{2}$$

$$C_{M+1,\pi(j),w(\pi(j))} + t_{\mathrm{u}} = C_{0,\pi(j+\rho),1} \quad \text{for } 1 \le j \le n-\rho, \tag{3}$$

$$C_{\max}(\pi) = C_{M+1,\pi(n),w(\pi(n))}. \tag{4}$$

Equation (1) expresses the completion time of a particular wafer at each stage i. The term $A_{i,l,t} + S_{i,l,\pi(j),k}$ denotes the time epoch when chamber l at stage i is ready for processing wafer k in job $\pi(j)$, and the term $C_{i-1,\pi(j),k}$ denotes the time epoch when the wafer is available to be processed at the chamber. Equation (2) describes the completion time of job $\pi(j)$ at stage $M+1$, where we assume that a waiting-for-process wafer in the port is at stage 0 and a finished wafer in the port is at stage $M+1$. Equation (3) expresses the job arrival/departure relationships for the dock. The equation indicates that job $\pi(j+\rho)$ in the WIP buffer can be transported to the dock only when job $\pi(j)$ in the dock has been moved away. Equation (4) computes the makespan $C_{\max}(\pi)$.

5. Genetic algorithm

A genetic algorithm (GA) is proposed to solve the in-line stepper scheduling problem. In the GA, a chromosome (a job sequence) is denoted by $\pi = [\pi(1), \ldots, \pi(n)]$ where $\pi(j)$ is called a *gene*, representing the job in the jth position of sequence π. The *makespan* $C_{\max}(\pi)$ of the chromosome, which can be calculated according to Section 4, is called its fitness function. The GA is designed to find a near-optimal solution from the huge solution space, which has $n!$ job sequences.

The GA was developed based on several prior genetic algorithms (Holland 1975, Gen and Cheng 2000). Compared with prior work, our GA is more comprehensive in the use of genetic operators with the inclusion of seven genetic operators. We first present the outline of the logic flow of the GA and proceed to explain the genetic operators.

Procedure of the GA

- Step 1: Set $t=0$. Randomly select N_{p} chromosomes to form an initial population $P(t)$.
- Step 2: Use crossover operators to create a set $N_1(t)$ of $P_{\mathrm{cr}} \cdot N_{\mathrm{p}}$ new chromosomes.
- Step 3: Use mutation operators to create a set $N_2(t)$ of $P_{\mathrm{m}} \cdot N_{\mathrm{p}}$ new chromosomes.
- Step 4: Set $S(t) = P(t) \cup N_1(t) \cup N_2(t)$. Use a selection strategy to select N_{p} chromosomes from $S(t)$ to form $P(t+1)$.
- Step 5: If a terminating condition is satisfied, stop. Otherwise, $t=t+1$ and go back to Step 2.

5.1 *Crossover operators*

To perform the crossover operation, we randomly select $(P_{cr} \cdot N_p)/2$ pairs of chromosomes from $P(t)$, where $0 < P_{cr} < 1$. These selected pairs are evenly divided into four groups. Each group is assigned a unique crossover operator from the following four: C1 (one point crossover) by Reeves (1995), LOX (linear order crossover) by Croce *et al.* (1995), PMX (partially mapped crossover) by Goldberg (1989), and the NABEL operator by Bac and Perov (1993). Of the four crossover operators, applying any one on a pair of chromosomes (called parents) will yield two new chromosomes (called children). As a result, a total of $P_{cr} \cdot N_p$ new chromosomes can be created. Each of the four crossover operators is explained below, where the two parent chromosomes are called *parent-1* and *parent-2*, and the two newly created chromosomes are called *child-1* and *child-2*.

5.1.1 *C1 operator*

Referring to Figure 3(a), one randomly selected point is used to divide each parent chromosome into two sections (*head-section* and *tail-section*). To create an offspring,

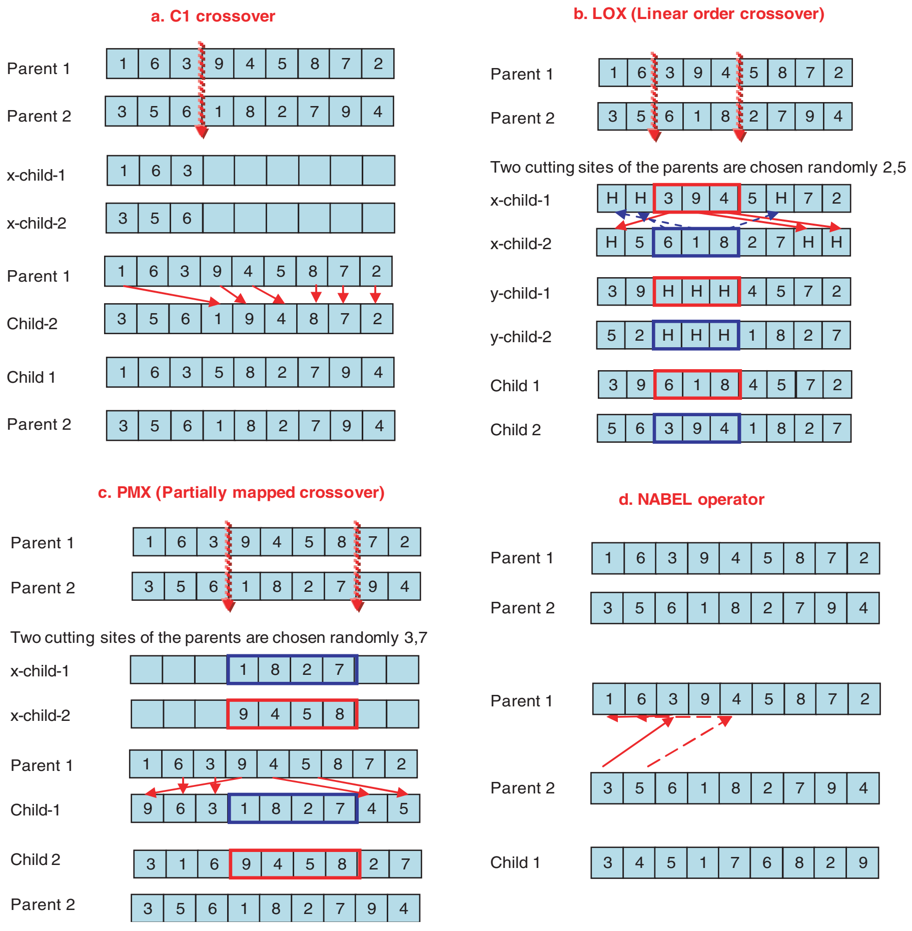
a. C1 crossover
Parent 1 1 6 3 9 4 5 8 7 2
Parent 2 3 5 6 1 8 2 7 9 4
x-child-1 1 6 3
x-child-2 3 5 6
Parent 1 1 6 3 9 4 5 8 7 2
Child-2 3 5 6 1 9 4 8 7 2
Child 1 1 6 3 5 8 2 7 9 4
Parent 2 3 5 6 1 8 2 7 9 4
b. LOX (Linear order crossover)
Parent 1 1 6 3 9 4 5 8 7 2
Parent 2 3 5 6 1 8 2 7 9 4
Two cutting sites of the parents are chosen randomly 2,5
x-child-1 H H 3 9 4 5 H 7 2
x-child-2 H 5 6 1 8 2 7 H H
y-child-1 3 9 H H H 4 5 7 2
y-child-2 5 2 H H H 1 8 2 7
Child 1 3 9 6 1 8 4 5 7 2
Child 2 5 6 3 9 4 1 8 2 7
c. PMX (Partially mapped crossover)
Parent 1 1 6 3 9 4 5 8 7 2
Parent 2 3 5 6 1 8 2 7 9 4
Two cutting sites of the parents are chosen randomly 3,7
x-child-1 1 8 2 7
x-child-2 9 4 5 8
Parent 1 1 6 3 9 4 5 8 7 2
Child-1 9 6 3 1 8 2 7 4 5
Child 2 3 1 6 9 4 5 8 2 7
Parent 2 3 5 6 1 8 2 7 9 4
d. NABEL operator
Parent 1 1 6 3 9 4 5 8 7 2
Parent 2 3 5 6 1 8 2 7 9 4
Parent 1 1 6 3 9 4 5 8 7 2
Parent 2 3 5 6 1 8 2 7 9 4
Child 1 3 4 5 1 7 6 8 2 9

Figure 3. Crossover operators: (a) C1, (b) PMX, (c) LOX, (d) NABEL.

for example *child-2*, its *head-section* is copied from that of *parent-2* and a string $(3, 5, 6)$ is obtained. Its tail-section is determined by referring to the job sequence of *parent-1* and considering the gene value eligibility. This implies that only the gene values not appearing in the head-section of *child-2* are eligible to appear in the tail-section. This results in a string $(1, 9, 4, 8, 7, 2)$ as the tail-section of *child-2*.

5.1.2 *LOX operator*

Referring to Figure 3(b), two randomly selected points are used to divide each parent chromosome into three sections (*head-section*, *mid-section*, and *tail-section*). In the figure, the mid-section for *parent-1* and *parent-2* is $(3, 9, 4)$ and $(6, 1, 8)$, respectively. A chromosome *x-child-1* is created by the first copying *parent-1* and by replacing by 'H' the gene values that appear in the mid-section of *parent-2*. This yields *x-child-1*$=(\mathrm{H}, \mathrm{H}, \underline{3, 9, 4}, 5, \mathrm{H}, 7, 2)$. By manipulating *x-child-1* through moving 'H' to the mid-section while keeping the sequence of the other gene values, we obtain a chromosome *y-child-1*$=(3, 9, \underline{\mathrm{H}, \mathrm{H}, \mathrm{H}}, 4, 5, 7, 2)$. Finally, replacing the mid-section of *y-child-1* with that of *parent-2*, we obtain an offspring *child-1*$=(3, 9, \underline{6, 1, 8}, 4, 5, 7, 2)$. The other offspring *child-2* can be obtained likewise.

5.1.3 *PMX operator*

Referring to Figure 3(c), two randomly selected points are used to divide each parent chromosome into three sections (head-section, mid-section, and tail-section). A new offspring (e.g., *child-1*) is created by the following procedure. The mid-section of *child-1* is created by referring to that of *parent-2*, which is a string $(1, 8, 2, 7)$. Both the head-section and tail-section of *child-1* are created by referring to *parent-1*. If the gene values in the head/tail sections of *parent-1* do not appear in *child-1*, we copy them in the exact positions of *child-1* (e.g., gene values 6 and 3). Finally, for those *vacant* genes in *child-1*, we place their values by sequentially referring to the *unassigned* genes in *parent-1*. This yields *child-1*$=(9, 6, 3, \underline{1, 8, 2, 7}, 4, 5)$.

5.1.4 *NABEL operator*

Suppose the two parent chromosomes are $\pi_{\mathrm{p1}} = [\pi_{\mathrm{p1}}(1), \ldots, \pi_{\mathrm{p1}}(n)]$ and $\pi_{\mathrm{p2}} = [\pi_{\mathrm{p2}}(1), \ldots, \pi_{\mathrm{p2}}(n)]$ and the two children chromosomes to be created are $\pi_{\mathrm{c1}} = [\pi_{\mathrm{c1}}(1), \ldots, \pi_{c1}(n)]$ and $\pi_{\mathrm{c2}} = [\pi_{\mathrm{c2}}(1), \ldots, \pi_{\mathrm{c2}}(n)]$. Referring to Figure 3(d), the NABEL operator is designed to set $\pi_{\mathrm{c1}}(i) = \pi_{\mathrm{p1}}(\pi_{\mathrm{p2}}(i))$ and $\pi_{\mathrm{c2}}(i) = \pi_{\mathrm{p2}}(\pi_{\mathrm{p1}}(i))$. For example, $\pi_{\mathrm{p2}}(2) = 5$ and $\pi_{\mathrm{p1}}(\pi_{\mathrm{p2}}(2)) = 4$, and we can obtain $\pi_{\mathrm{c1}}(2) = \pi_{\mathrm{p1}}(\pi_{\mathrm{p2}}(2)) = 4$.

5.2 *Mutation operators*

Mutation operators are often incorporated in a GA to avoid early convergence into local optima. In performing mutation operations, we randomly select $P_{\mathrm{mu}} \cdot N_{\mathrm{p}}$ number of chromosomes from $P(t)$, and evenly divide them into three groups. Each group is manipulated by a particular type of the following three mutation operators – Swap, Inverse, and Insert (Wang and Zheng 2003). In a mutation operation, each selected chromosome (π_{a}) is used to generate a new chromosome (π_{b}).

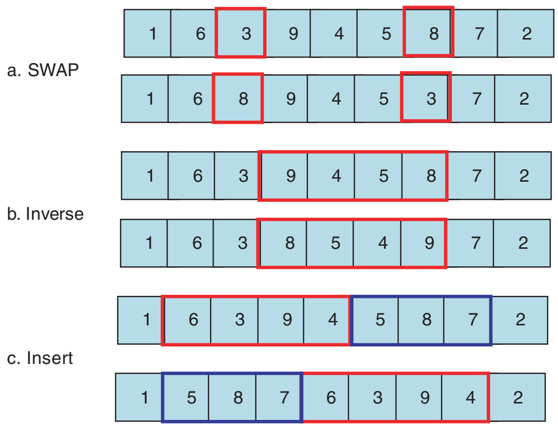

Figure 4. Mutation operators: (a) SWAP, (b) Inverse, (c) Insert.

5.2.1 *Swap operator*

Referring to Figure 4(a), we randomly choose two distinct genes in π_a, and then swap their gene values to create π_b.

5.2.2 *Inverse operator*

Referring to Figure 4(b), we randomly select two cut-off points in π_a and divide them into three sections. Denote $\pi_a = \{\pi_{a1}, \pi_{a2}, \pi_{a3}\}$ and π_{a2}=[$\pi_{a2}(1), \ldots, \pi_{a2}(m)$]. The inverse operator is designed to create a new chromosome $\pi_b = \{\pi_{a1}, \pi_{b2}, \pi_{a3}\}$, where $\pi_{b2}(i) = \pi_{a2}(m + 1 - i)$.

5.2.3 *Insert operator*

Referring to Figure 4(c), we randomly select an *insert point* and a *segment* of genes in π_a. This would divide π_a into four sections $\pi_a = \{\pi_{a1} | \pi_{a2}, \boxed{\pi_{a3}}, \pi_{a4}\}$, which indicates that the insert point is between π_{a1} and π_{a2}, and the selected segment is π_{a3}. To create a new chromosome, the insert operator moves the selected segment to the insert point position. This would yield a new chromosome $\pi_b = \{\pi_{a1}, \boxed{\pi_{a3}} \| \pi_{a2}, \pi_{a4}\}$.

5.3 ***Selection strategy***

After completing the genetic operations, the chromosomes in $P(t)$ and the newly created chromosomes are $Q = N_p(1 + P_{cr} + P_{mu})$ in number and only N_p of them are to be selected to form $P(t+1)$. To do so, we first sort the Q chromosomes in descending order, in terms of fitness values. Let the sorted chromosomes be represented by $\pi_1, \ldots, \pi_Q$. When forming $P(t+1)$, where π_1 is always selected and the other π_i are selected based on probability – the roulette wheel selection method is applied (Goldberg 1989, Michalewicz 1996).

5.4 ***Terminating conditions***

There are two termination conditions. First, the GA terminates when the best solution in $P(t)$ is the same for over N generations. Second, the GA is forcedly terminated when $t = Y$, where Y is a large positive integer.

6. Numerical experiments

Using numerical experiments, we attempt to demonstrate the effectiveness of the proposed GA. Assumptions concerning the configuration and operations of the in-line stepper are presented below. The in-line stepper, equipped with four ports, has 14 stages and 21 chambers. Moreover, the operation time at each chamber i follows a uniform distribution $[a_i, b_i]$ (Table 1). A mask setup is always required for the aligner chamber while it turns to process a new job's wafer, and the mask change time is a constant (1.0 min).

6.1 *Design of test scenarios*

To model various process yield scenarios, we use four job-size distributions: the *truncated binomial distribution*, uniformly distributed, linearly increasing, and linearly decreasing. The shapes of the *truncated binomial distribution* may be convex or concave. Thus, all possible types of distribution shapes (convex, concave, uniform, decreasing, and increasing) have been considered in the tests. We use (N, Y) to represent the number of test examples, where N represents the number of jobs to be scheduled and Y represents the number of process yield distributions.

For each test example, we compare the GA with four benchmark scheduling rules, namely Random, SPT (shortest job processing time), LPT (longest job processing time), and NEH (Nawaz *et al.* 1983). The sequencing rule Random denotes that the job sequence is determined by random selection. For Random, the average performance of 100 experimental runs is counted. For GA, the average performance of 15 experimental runs is counted. The GA is implemented by setting $P = 100$, $P_{\text{cr}} = 0.6$, $P_{\text{mu}} = 0.2$, and $N = 5000$.

Define C_{G} as the makespan of the GA, C_{random}, C_{SPT}, C_{LPT} and C_{NEH} as the makespans of the four benchmark methods, and $C_{\text{B}} = \min\{C_{\text{Random}}, C_{\text{SPT}}, C_{\text{LPT}}, C_{\text{NEH}}\}$. The metric $\gamma_{\text{G}} = (C_{\text{B}} - C_{\text{G}})/C_{\text{B}}$ is used to measure the effectiveness of the GA. The larger γ_{G}, the better the GA.

6.2 *Experimental results*

The truncated binomial distribution (TBD) for yield modeling implies that the job size is governed by a *binomial distribution*; however, the jobs carrying no wafers are moved away from the fab. In the TBD yield scenario, 50 examples ($N = 5$, $Y = 10$) are tested. As shown in Table 2, the GA outperforms the four benchmarks in each of the 50 test examples, with γ_{G} ranging from 0.23 to 12.41%.

The uniform distribution (UNF) for yield modeling is represented by $U[1, n]$, which denotes that the probability for a job with k $(1 \le k \le n)$ good wafer sizes is $1/n$. In the UNF yield scenario, 25 examples ($N = 5$, $Y = 5$) are tested. The five uniform distributions ($Y = 5$) include $U[1, 25]$, $U[1, 20]$, $U[1, 15]$, $U[1, 10]$, and $U[1, 5]$. See Table 3, where it is shown that the GA outperforms the four benchmarks in each of the 25 test examples, with γ_{G} ranging from 0.65 to 9.3%.

The linearly increasing/decreasing distribution for yield modeling is designed by setting different percentages on the five particular job sizes. In this yield scenario, 10 examples ($N = 5$, $Y = 2$) are tested. As shown in Table 4, the GA also outperforms the four benchmarks in each of the 10 test examples, with γ_{G} ranging from 0.28 to 1.33%.

The experimental results indicate that the proposed GA has its merits, particularly for a wafer fab that includes a substantial number of small-sized jobs. With in-line steppers

Table 1. Process times of in-line stepper chambers.

Process sequence	WIP buffers to dock area	Dock area to track	HMDS	Cooling	Coater	Soft bake	Cooling	Aligner	Wafer edge exposure	PEB	Cooling	Develop	Hard bake	High cooling
Chamber number	1	1	2	2	2	2	2	1	1	2	2	2	2	1
Process time (min)	2.5	0.1	1.2	1.2	1.2	[1.2, 2.8]	1	[0.75, 1.65]	1	[1.2, 2.8]	1	[1.2, 2.8]	[1.2, 2.8]	0.5

Table 2. Makespan comparison for different binomial yield scenaries.

Jobs	20							40							60							80							100						
	Makespan							Makespan							Makespan							Makespan							Makespan						
Yield	Random (min)	SPT (min)	LPT (min)	NEH (min)	C_B (min)	C_G (min)	γ_G	Random (min)	SPT (min)	LPT (min)	NEH (min)	C_B (min)	C_G (min)	γ_G	Random (min)	SPT (min)	LPT (min)	NEH (min)	C_B (min)	C_G (min)	γ_G	Random (min)	SPT (min)	LPT (min)	NEH (min)	C_B (min)	C_G (min)	γ_G	Random (min)	SPT (min)	LPT (min)	NEH (min)	C_B (min)	C_G (min)	γ_G
90%	572.6	571.6	565.1	565.2	565.1	561.8	0.59%	1170.0	1170.6	1158.7	1160.6	1158.7	1154.8	0.33%	1726.0	1730.6	1710.2	1714.3	1710.2	1706.3	0.23%	2297.0	2305.3	2277.7	2282.8	2277.7	2273.2	0.20%	2853.9	2862.2	2832.4	2837.3	2832.4	2826.0	0.23%
80%	539.9	537.1	532.9	534.2	532.9	529.6	0.62%	1015.7	1011.9	1004.4	1008.0	1004.4	1000.7	0.37%	1521.8	1519.5	1508.0	1510.8	1508.0	1502.9	0.34%	2056.4	2055.0	2038.8	2043.7	2038.8	2034.3	0.22%	2564.6	2563.0	2540.8	2547.1	2540.8	2537.2	0.14%
70%	458.0	454.2	450.9	453.1	450.9	447.7	0.70%	914.7	910.7	905.4	906.8	905.4	900.5	0.55%	1341.1	1335.8	1328.1	1329.8	1328.1	1322.8	0.41%	1827.3	1818.1	1809.8	1813.3	1809.8	1805.0	0.27%	2282.2	2273.1	2260.5	2267.2	2260.5	2256.2	0.19%
60%	400.5	398.4	395.1	394.4	394.4	390.5	1.00%	787.9	781.5	778.3	781.4	778.3	773.4	0.63%	1210.2	1201.3	1197.3	1199.5	1197.3	1191.9	0.45%	1561.2	1547.4	1544.4	1547.7	1544.4	1538.8	0.37%	1961.2	1946.5	1941.8	1946.1	1941.8	1935.6	0.32%
50%	359.1	356.5	353.3	353.6	353.3	349.6	1.05%	691.0	685.7	681.8	683.1	681.8	677.0	0.70%	977.2	967.4	964.8	967.3	964.8	959.8	0.52%	1351.3	1338.9	1335.0	1337.7	1335.0	1329.3	0.43%	1608.7	1593.5	1589.2	1593.5	1589.2	1582.6	0.42%
40%	277.0	276.6	272.4	272.6	272.4	267.7	1.72%	531.9	527.2	523.8	525.7	523.8	518.4	1.04%	811.1	801.4	798.6	800.5	798.6	793.8	0.60%	1061.9	1053.1	1050.9	1050.0	1050.0	1040.2	0.93%	1357.6	1342.0	1338.1	1341.3	1338.1	1331.3	0.51%
30%	224.8	222.9	220.0	219.2	219.2	215.2	1.85%	419.0	413.8	411.3	411.9	411.3	405.6	1.40%	586.4	578.0	575.5	577.2	575.5	564.4	1.93%	855.2	844.0	841.5	843.5	841.5	834.1	0.88%	1039.1	1023.4	1017.4	1022.7	1017.4	1009.3	0.80%
25%	174.9	170.9	168.9	168.5	168.5	162.3	3.69%	367.3	358.8	356.1	357.3	356.1	346.2	2.80%	502.6	486.1	482.2	489.4	482.2	468.7	2.82%	703.1	688.1	684.3	685.3	684.3	670.9	1.96%	929.1	911.0	908.9	911.0	908.9	896.4	1.38%
20%	159.2	150.1	151.3	149.2	149.2	134.9	9.55%	332.8	319.7	319.3	314.4	314.4	270.3	14.03%	462.3	445.8	444.1	452.3	444.1	416.0	6.33%	639.0	616.3	614.9	625.5	614.9	585.7	4.74%	792.8	759.7	759.3	770.0	759.3	734.9	3.22%
15%	146.3	143.4	141.0	139.7	139.7	122.4	12.41%	290.7	268.2	267.2	272.9	267.2	251.5	5.89%	454.9	400.2	399.4	397.2	397.2	360.2	9.31%	727.8	517.1	517.2	525.1	517.1	457.5	11.52%	711.5	648.5	646.9	652.7	646.9	607.3	6.13%

Table 3. Makespan comparison for uniform yield distributions.

Jobs	20							40							60							80							100						
	Makespan							Makespan							Makespan							Makespan							Makespan						
Uniform	Random (min)	SPT (min)	LPT (min)	NEH (min)	C_B (min)	C_G (min)	γ_G	Random (min)	SPT (min)	LPT (min)	NEH (min)	C_B (min)	C_G (min)	γ_G	Random (min)	SPT (min)	LPT (min)	NEH (min)	C_B (min)	C_G (min)	γ_G	Random (min)	SPT (min)	LPT (min)	NEH (min)	C_B (min)	C_G (min)	γ_G	Random (min)	SPT (min)	LPT (min)	NEH (min)	C_B (min)	C_G (min)	γ_G
U[1,25]	340.9	341.5	335.2	336.4	335.2	331.1	1.22%	667.8	663.9	661.6	657.4	657.4	652.7	0.71%	1057.9	1057.2	1052.4	1052.3	1052.3	1037.7	1.39%	1383.0	1389.2	1382.9	1365.1	1365.1	1356.3	0.65%	1672.8	1671.3	1669.3	1654.8	1654.8	1643.2	0.70%
U[1,20]	310.5	311.2	306.9	304.5	304.5	300.9	1.18%	583.6	575.0	574.8	574.6	574.6	564.1	1.83%	885.9	885.2	880.4	873.5	873.5	866.0	0.86%	1107.1	1103.4	1099.6	1089.0	1089.0	1073.4	1.43%	1394.3	1386.5	1399.1	1366.7	1366.7	1357.6	0.66%
U[1,15]	213.9	209.6	208.2	200.0	200.0	194.3	2.85%	453.2	442.4	443.5	438.0	438.0	431.7	1.45%	642.2	631.1	628.4	621.0	621.0	610.9	1.62%	877.1	860.0	860.8	862.4	860.0	835.9	2.81%	1022.2	1002.0	998.9	983.8	983.8	957.7	2.65%
U[1,10]	173.6	168.0	168.4	158.9	158.9	155.2	2.34%	330.1	317.4	316.9	318.7	316.9	303.2	4.33%	517.8	503.9	504.1	499.1	499.1	478.0	4.23%	620.0	599.6	598.2	593.9	593.9	538.6	9.30%	811.9	786.3	787.0	790.0	786.3	749.8	4.65%
U[1,5]	131.5	123.0	124.9	124.8	123.0	116.6	5.16%	262.9	250.4	250.3	247.6	247.6	232.9	5.95%	377.1	354.5	358.2	362.0	354.5	341.4	3.72%	487.9	466.4	464.7	475.9	464.7	448.0	3.59%	622.9	587.7	589.6	596.0	587.7	572.1	2.65%

Table 4. Makespan comparison for linearly increasing/decreasing yield distribution.

Jobs	20							40							60							80							100						
	Makespan							Makespan							Makespan							Makespan							Makespan						
Uniform	Random (min)	SPT (min)	LPT (min)	NEH (min)	C_B (min)	C_G (min)	γ_G	Random (min)	SPT (min)	LPT (min)	NEH (min)	C_B (min)	C_G (min)	γ_G	Random (min)	SPT (min)	LPT (min)	NEH (min)	C_B (min)	C_G (min)	γ_G	Random (min)	SPT (min)	LPT (min)	NEH (min)	C_B (min)	C_G (min)	γ_G	Random (min)	SPT (min)	LPT (min)	NEH (min)	C_B (min)	C_G (min)	γ_G
U[1,25]	340.9	341.5	335.2	336.4	335.2	331.1	1.22%	667.8	663.9	661.6	657.4	657.4	652.7	0.71%	1057.9	1057.2	1052.4	1052.3	1052.3	1037.7	1.39%	1383.0	1389.2	1382.9	1365.1	1365.1	1356.3	0.65%	1672.8	1671.3	1669.3	1654.8	1654.8	1643.2	0.70%
U[1,20]	310.5	311.2	306.9	304.5	304.5	300.9	1.18%	583.6	575.0	574.8	574.6	574.6	564.1	1.83%	885.9	885.2	880.4	873.5	873.5	866.0	0.86%	1107.1	1103.4	1099.6	1089.0	1089.0	1073.4	1.43%	1394.3	1386.5	1399.1	1366.7	1366.7	1357.6	0.66%
U[1,15]	213.9	209.6	208.2	200.0	200.0	194.3	2.85%	453.2	442.4	443.5	438.0	438.0	431.7	1.45%	642.2	631.1	628.4	621.0	621.0	610.9	1.62%	877.1	860.0	860.8	862.4	860.0	835.9	2.81%	1022.2	1002.0	998.9	983.8	983.8	957.7	2.65%
U[1,10]	173.6	168.0	168.4	158.9	158.9	155.2	2.34%	330.1	317.4	316.9	318.7	316.9	303.2	4.33%	517.8	503.9	504.1	499.1	499.1	478.0	4.23%	620.0	599.6	598.2	593.9	593.9	538.6	9.30%	811.9	786.3	787.0	790.0	786.3	749.8	4.65%
U[1,5]	131.5	123.0	124.9	124.8	123.0	116.6	5.16%	262.9	250.4	250.3	247.6	247.6	232.9	5.95%	377.1	354.5	358.2	362.0	354.5	341.4	3.72%	487.9	466.4	464.7	475.9	464.7	448.0	3.59%	622.9	587.7	589.6	596.0	587.7	572.1	2.65%

as the bottleneck of a fab, even a 1% increase in the in-line stepper throughput would have a substantial positive impact on gross margins.

7. Concluding remarks

This study examines a scheduling problem for a semiconductor in-line stepper operating in a low-yield environment. The scheduling problem, rarely studied in the earlier literature, is essentially a parallel-machine flow shop scheduling problem with four salient features: *mask setup*, *port-capacity constraint*, *dock–stepper transportation incompatibility*, *and variable job size*. In-line steppers are the usual bottlenecks of wafer fabs, thus their scheduling significantly affects the ultimate throughput of a wafer fab.

We propose a GA method in order to minimise the makespan. The GA includes seven genetic operators: four crossover and three mutation operators. Numeric experiments were extensively tested. The experimental results indicate that the GA outperforms the four benchmark scheduling rules, particularly in an NPI scenario that, by its very nature, would involve a substantial number of small-sized jobs.

Possible extensions of this research include the development of other meta-heuristic algorithms such as tabu search, ant colony, and particle swarm methods. Effective heuristic scheduling methods that are computationally efficient may also be considered.

References

Allahverdi, A., *et al*., 1999. A review of scheduling research involving setup considerations. *OMEGA*, 27 (2), 219–239.

Aytug, H., *et al*., 1994. A review of machine learning in scheduling. *IEEE Transactions on Engineering Management*, 41 (2), 165–171.

Bac, F.Q. and Perov, V.L., 1993. New evolutionary genetic algorithms for NP-complete combinatorial optimization problems. *Biological Cybernetics*, 69 (3), 229–234.

Bagchi, T.P., 2006. A review of TSP based approaches for flowshop scheduling. *European Journal of Operational Research*, 169 (3), 816–854.

Cavalieri, S., Crisafulli, F., and Mirabella, O., 1999. A genetic algorithm for job-shop scheduling in a semiconductor manufacturing system. *IEEE Transactions on Engineering Management*, 41 (2), 957–961.

Cheng, T.C.E., Jatinder, N.D., and Wang, G.G., 2000. A review of flowshop scheduling research with setup times. *Production and Operations Management*, 9 (3), 262–282.

Chern, C.C. and Liu, Y.L., 2003. Family-based scheduling rules of a sequence-dependent wafer fabrication system. *IEEE Transactions on Semiconductor Manufacturing*, 16 (1), 15–25.

Croce, F.D., Tadei, R., and Volta, G., 1995. A genetic algorithm for the job shop problem. *Computers in Operational Research*, 22 (1), 15–24.

Dabbas, R.M. and Fowler, J.W., 2003. A new scheduling approach using combined dispatching criteria in wafer fabs. *IEEE Transactions on Semiconductor Manufacturing*, 16 (3), 501–510.

Danneberg, D., Tautenhahn, T., and Werner, F., 1998. A comparison of heuristic algorithms for flow shop scheduling problems with set-up times and limited batch size. Preprint, Otto-von-Guericke University, Magdeburg.

Diaz, S.L., *et al*., 2005. Evaluating the impacts of reticle requirements in semiconductor wafer fabrication. *IEEE Transactions on Semiconductor Manufacturing*, 18 (4), 622–632.

Duwayri, Z., Mollaghasemi, D.N., and Rabadi, G., 2006. Scheduling setup changes at bottleneck workstations in semiconductor manufacturing. *Production Planning & Control*, 17 (7), 717–727.

Gen, M. and Cheng, R., 2000. *Genetic algorithms and engineering optimization*. New York: Wiley.

Goldberg, D.E., 1989. *Genetic algorithms in search, optimization and machine learning*. Boston: Addison-Wesley.

Graham, R.L., *et al*., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5, 287–326.

Gupta, A.K. and Sivakumar, A.I., 2006. Jop shop scheduling techniques in semiconductor manufacturing. *International Journal of Advanced Manufacturing Technology*, 27 (11-12), 1163–1169.

Hejazi, S.R. and Saghafian, S., 2005. Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research*, 43 (14/15), 2895–2929.

Holland, J.H., 1975. *Adaptation in neural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Kang, K.H. and Lee, Y.H., 2007. Make-to-order scheduling in foundry semiconductor fabrication. *International Journal of Production Research*, 45 (3), 615–630.

Michalewicz, Z., 1996. *Genetic algorithms+data structures=evolution programs*. 3rd ed. Berlin: Springer.

Miwa, T., Nishihara, N., and Yamamoto, K., 2005. Automated stepper load balance allocation system. *IEEE Transactions on Semiconductor Manufacturing*, 18 (4), 510–516.

Morrison, J.R. and Martin, D.P., 2007. Performance evaluation of photolithography cluster tools. *OR Spectrum*, 29 (3), 375–389.

Nawaz, M., Enscore Jr., E.E., and Ham, I., 1983. A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem. *OMEGA, The International Journal of Management Science*, 11 (1), 91–95.

Panwalker, S. and Iskander, W., 1977. A survey of scheduling rules. *Operations Research*, 25 (1), 45–61.

Pinedo, M., 2000. *Scheduling – theory, algorithms, and systems*. 2nd ed. New York: Prentice Hall.

Pranzo, M., 2001. Batch scheduling in a two machine flowshop with limited buffer and sequence independent setup times. *In*: *Proceedings of the ORP3*, 26–29 September 2001, Paris, 1–6.

Quadt, D. and Kuhn, H., 2007. A taxonomy of flexible flow line scheduling procedures. *European Journal of Operational Research*, 178 (3), 686–698.

Quirk, M., 2001. *Semiconductor manufacturing technology*. New York: Prentice Hall.

Reeves, C.R., 1995. A genetic algorithm for flowshop sequencing. *Computers in Operational Research*, 22 (1), 5–13.

Reisman, A., Kumar, A., and Motwani, J., 1997. Flowshop scheduling/sequencing research: a statistical review of the literature, 1952–1994. *IEEE Transactions on Engineering Manufacturing*, 44 (3), 316–329.

Ruiz, R. and Maroto, C., 2006. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *European Journal of Operational Research*, 169 (3), 781–800.

Sha, D.Y., *et al*., 2006. A dispatching rule for photolithography scheduling with an on-line rework strategy. *Computers & Industrial Engineering*, 50 (3), 233–247.

Sriskandarajah, C., 1993. Performance of scheduling algorithms for no-wait flowshops with parallel machines. *European Journal of Operational Research*, 70 (3), 365–378.

Wang, C.S. and Uzsoy, R., 2002. A genetic algorithm to minimize maximum lateness on a batch processing machine. *Computers in Operational Research*, 29 (12), 1621–1640.

Wang, L. and Zheng, D.Z., 2003. An effective hybrid heuristic for flowshop scheduling. *International Journal of Advanced Manufacturing Technology*, 21 (1), 38–44.

Wu, M.C., *et al*., 2006. Dispatching in semiconductor fabs with machine-dedication features. *International Journal of Advance Manufacturing Technology*, 28 (9-10), 978–984.

Wu, M.C., Chiou, S.J., and Chen, C.F., 2008a. Dispatching for make-to-order wafer fabs with machine-dedication and mask set-up characteristics. *International Journal of Production Research*, 46 (14), 3993–4009.

Wu, M.C., Jiang, J.H., and Chang, W.J., 2008b. Scheduling a hybrid MTO/MTS semiconductor fab with machine-dedication features. *International Journal of Production Economics*, 112 (1), 416–426.

Xiao, H., 2001. *Introduction to semiconductor manufacturing technology*. New York: Prentice Hall.

Yang, W.H. and Liao, C.J., 1999. Survey of scheduling research involving setup times. *International Journal of Systems Science*, 30 (2), 143–155.

Zant, P.V., 2004. *Microchip fabrication*. New York: McGraw-Hill.

Zhu, X. and Wilhelm, W.E., 2006. Scheduling and lot sizing with sequence-dependent setup: a literature review. *IIE Transactions*, 38 (11), 987–1007.